2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Boosting Exploration in Actor-Critic Algorithms by Incentivizing Plausible Novel States

Chayan Banerjee, Zhiyong Chen, and Nasimul Noman

Abstract— **Improvement of exploration and exploitation using more efficient samples is a critical issue in reinforcement learning algorithms. A basic strategy of a learning algorithm is to facilitate indiscriminate exploration of the entire environment state space, as well as to encourage exploration of rarely visited states rather than frequently visited ones. Under this strategy, we propose a new method to boost exploration through an intrinsic reward, based on the measurement of a state's novelty and the associated benefit of exploring the state, collectively called plausible novelty. By incentivizing exploration of plausible novel states, an actor-critic (AC) algorithm can improve its sample efficiency and, consequently, its training performance. The new method is verified through extensive simulations of continuous control tasks in MuJoCo environments, using a variety of prominent off-policy AC algorithms.**

I. INTRODUCTION

Reinforcement learning (RL) algorithms have achieved state-of-the-art performance for many control problems. A typical strategy of a model-free RL algorithm is to train an agent by explicitly learning a policy network aided by a concurrently learned state-value (V-value) or action-value (Q-value) network, within the actor-critic (AC) architecture [1]. To boost sample efficiency, an off-policy RL algorithm maintains a so-called experience replay (ER) buffer, which stores all past samples for future reuse. In an off-policy AC algorithm, actor and/or critic networks are trained using data uniformly sampled from an ER buffer.

Among prominent off-policy AC algorithms, a deep deterministic policy gradient algorithm (DDPG) [2], [3] trains a deterministic policy network and a Q-value network (or simply called Q-network) and it encourages exploration in action space by simply adding noise to actions. Although DDPG can achieve superior performance in certain environments, it suffers from the issue of overestimating Q-values. A twin delayed DDPG (TD3) [4] algorithm was introduced to mitigate the overestimation issue using multiple innovations including delayed policy-update, target policy smoothing, and a clipped double Q-value learning approach. As an ER buffer in off-policy algorithms needs a large number of samples to maintain a population for meaningful policy learning, researchers studied a variety of exploration strategies to improve sampling efficiency, such as action space perturbation used in DDPG and TD3, and policy parameter perturbation in [5], [6]. These strategies have their features in different environments, but they do no always perform well in high dimensional and sparse reward environments. It motivates us to further study more advanced strategies for boosting exploration in off-policy AC algorithms.

The basic idea used in this paper is to boost exploration by adding exploration bonus, or called an intrinsic reward, to a reward function. Specifically, an original extrinsic reward r_t is combined with an intrinsic reward r_t^{intr} to form an augmented reward r_t^{aug}, denoted by

$$r_t^{\mathrm{aug}} = r_t \oplus r_t^{\mathrm{intr}} \tag{1}$$

where $\oplus$ represents aggregation between the two sources of rewards. The intrinsic reward fundamentally quantifies the novelty of a new state in an exploration process. A more novel state would receive a higher intrinsic reward and raise the probability of choosing an action to visit it.

There are extensive researches using the idea of intrinsic reward in the literature, although not in the off-policy AC architecture. An intrinsic reward can be defined based on state visitation count. For example, the intrinsic reward in [7] quantifies the count of visitations of a certain state using a density model and a pseudo-count generating method. In [8], a variational Gaussian mixture model is used to estimate densities of trajectories, that is, counts of visitations to a sequence of states and actions. In [9], occurrences of high dimensional states are recorded and mapped into discrete hash codes to form intrinsic rewards.

An intrinsic reward can also be calculated based on a prediction error method; an exploration bonus is awarded if there is improvement of an agent's knowledge about the environment dynamics through a predictive model. The authors of [10] proposed a forward dynamics model that is trained in encoded state space using an autoencoder. A state's novelty and hence the intrinsic reward of visiting the state is calculated based on the model's prediction error with respect to the state. The authors of [11] introduced multiple forward dynamics models and used the variance over the model outputs as an intrinsic reward.

Another approach of obtaining an intrinsic reward is through maximizing entropy of actions, states, or state-action pairs. For instance, an objective function is augmented with an intrinsic reward based on a policy's entropy in [12]. The work in [13] uses a strategy of maximizing the Renyi entropy over state-action space for better exploration but in a reward free RL setting. The idea was further improved and extended for a reward based RL framework in [14]. It uses a proxy reward, which consists of an extrinsic reward from environment, augmented by an action entropy maximization

C. Banerjee and Z. Chen are with the School of Engineering, The University of Newcastle, Callaghan, NSW 2308, Australia. N. Noman is with the School of Information and Physical Sciences, The University of Newcastle, Callaghan, NSW 2308, Australia. Z. Chen is the corresponding author. Email: zhiyong.chen@newcastle.edu.au

reward, and an additional intrinsic reward that motivates state entropy maximization.

Now, an interesting question is how to fit intrinsic reward based strategies into the architecture of off-policy AC algorithms. On one hand, assigning a reward bonus solely based on a certain novelty value indiscriminately offered for a state, as seen in the aforementioned references, is insufficient for boosting exploration in AC algorithms, because some novel states may not be worth exploring if they have a poor chance of benefiting the policy optimization process. On the other hand, soft actor-critic (SAC) [15], [16] is one of the most efficient action entropy maximization based algorithms for benefiting a policy optimization process. Increase in policy's entropy results in more exploration and accelerates learning in SAC. A physics informed intrinsic reward is proposed in an AC algorithm in [17], which aims to assist an agent to overcome the difficulty of poor training when a reward function is sparse or misleading in short term. However, neither of them includes state novelty into the calculation of intrinsic rewards.

The new strategy proposed in this paper focuses on exploration towards the states that have higher chances of positively impacting policy optimization, based on the measurement of a state's novelty as well as the associated benefit of exploring the state, which is called a state's plausible novelty. To the best of our knowledge, this is the first attempt to consider both state novelty and benefit of exploring a state towards policy optimization in calculating an intrinsic reward. Another interesting feature of the new strategy is that it is an add-on/secondary artifact that can be applied to any primary off-policy AC type algorithm to improve its training performance. In particular, three state-of-the-art off-policy AC algorithms, SAC, DDPG, and TD3, are respectively treated as the primary algorithms in this paper.

The new strategy is called incentivizing plausible novel states (IPNS) throughout the paper and it brings two major innovations. First, we propose a new state novelty scoring scheme, based on estimating a high visitation density (HVD) point from past experience. The score quantifies the Euclidean distance between a current state and an HVD point. Second, we introduce a plausible novelty (PN) score which is a combined quantification of a state's novelty score and its chance of positive contribution towards policy optimization. The chance is estimated by its V-value predicted by a concurrently trained V-network. The PN score is then normalized and weighted as an intrinsic reward bonus to be added on a primary algorithm's extrinsic reward to form the final augmented reward, which is used in policy learning.

II. PRELIMINARIES AND MOTIVATION

We consider a Markovian dynamical system represented by a conditional probability density function $p(s_{t+1}|s_t, a_t)$, where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are the state and action, respectively, at time instant $t = 1, 2, \cdots$. Here, $\mathcal{S}$ and $\mathcal{A}$ represent continuous state and action spaces, respectively. Under a stochastic control policy $\pi_\phi(a_t|s_t)$, parameterized by ϕ, the distribution of the closed-loop trajectory $\tau =$

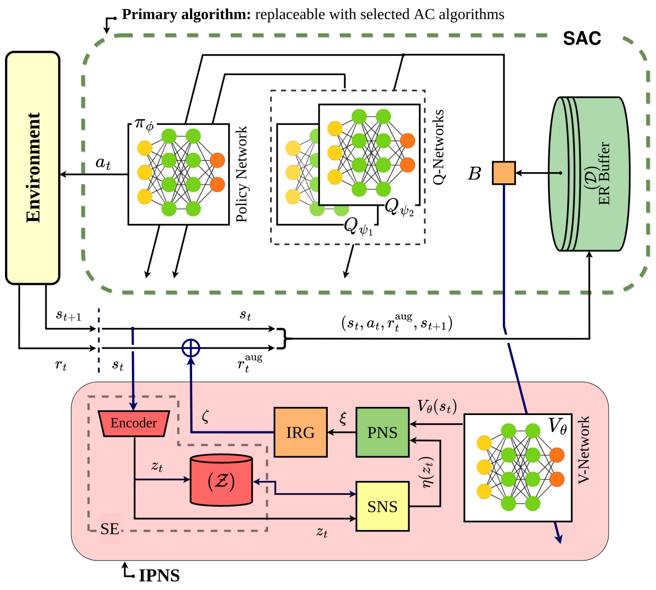

Fig. 1: Illustration of the proposed IPNS module paired with SAC as a primary algorithm.

$(s_1, a_1, s_2, a_2, \cdots, s_T, a_T, s_{T+1})$, over one episode $t = 1, \cdots, T$, can be represented by

$$p_\phi(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\phi(a_t|s_t) p(s_{t+1}|s_t, a_t).$$

Denote $r_t = R(a_t, s_{t+1})$ as the reward generated at time t. An optimal policy is represented by the parameter

$$\phi^* = \arg\max_\phi \underbrace{\mathbf{E}_{\tau \sim p_\phi(\tau)} \Big[\sum_{t=1}^{T} \gamma^t r_t \Big]}_{G_\phi},$$

which maximizes the objective function G_ϕ with a discount factor $\gamma \in (0, 1)$. The discounted future reward $\sum_{t=1}^{T} \gamma^t r_t$ is also called a return.

The main objective of this paper is to develop a new incentive mechanism that can be applied to boost exploration in off-policy AC algorithms. In this section, we use SAC as a primary AC algorithm to introduce the incentive mechanism. SAC uses a maximum entropy objective, formed by augmenting the typical RL objective G_ϕ with the expected entropy of the policy over $p_\phi(\tau)$. In other words, an agent receives an intrinsic reward at each time step which is proportional to the policy's entropy at that time-step, given by $r_t^{\text{intr}} = \mathcal{H}(\pi_\phi(\cdot|s_t))$. So, the SAC's entropy-regularized RL objective to find an optimal policy can be written as

$$\phi^* = \arg\max_\phi \mathbf{E}_{\tau \sim p_\phi(\tau)} \big[\sum_{t}^{T} \gamma^t \big(r_t + \alpha \mathcal{H}(\pi_\phi(\cdot|s_t)) \big) \big],$$

where the entropy regularization coefficient α determines the relative importance of the entropy term against the reward. The version with a constant α is used in this paper.

The architecture of SAC is illustrated in Fig. 1. SAC learns a policy π_ϕ in the policy network (actor), which takes in the current state and generates the mean and standard deviation of an action distribution (defining a Gaussian). It concurrently learns two Q-networks Q_{ψ_1}, Q_{ψ_2} (critic) to generate Q-value, which assesses the expected return of a pair of state and action. The Q-networks are learnt by regressing to the values generated by a shared pair of target networks, which are obtained by exponentially moving-averaging the Q-network parameters over the course of training.

As an off-policy algorithm, SAC alternates between a "data collection" phase and a "network parameter update" phase. In the data collection phase, SAC saves transition tuples $d_t^e = (s_t^e, a_t^e, r_t^e, s_{t+1}^e)$ to an ER buffer $\mathcal{D}$, for $t = 1, \cdots, T_e,\ e = 1, \cdots, E$ (E is total episodes run). The tuples are obtained by running the current policy in the environment. In the network parameter update phase, SAC uniformly samples a mini-batch (B) of saved transition tuples from the ER buffer $\mathcal{D}$ and updates the network parameters. The total timesteps of the policy learning process is $N = \sum_{e=1}^{E} T_e$.

The new IPNS mechanism for boosting exploration in an off-policy AC algorithm is through defining an intrinsic reward based on measurement of a state's novelty and the associated benefit of exploring the state, which altogether is called plausible novelty. IPNS consists of four functional modules: state encoder (SE), state novelty scorer (SNS), plausible novelty scorer (PNS), and Intrinsic reward generator (IRG). These four modules are illustrated in Fig. 1 and also elaborated in the next section with more details. A primary AC algorithm, with the extrinsic reward r_t replaced by the new augmented extrinsic/intrinsic reward r_t^{aug} following the IPNS mechanism, is enhanced to a new RL algorithm, whose advantages in boosting its exploration capacity will be examined in various benchmark environments.

III. INCENTIVIZING PLAUSIBLE NOVEL STATES

The specific design of the four functional modules of the proposed IPNS strategy is discussed in the following four subsections, respectively.

A. State encoder (SE)

The SE module includes an autoencoder and an encoded state buffer $\mathcal{Z}$. At each timestep, the encoder performs dimensionality reduction on the original state vector $s_t \in \mathbf{R}^m$ and extracts the latent structure of the state vector in the form of a compressed and normalized code $z_t \in \mathbf{R}^{m'}$ with $m' < m$. Dimensionality reduction is beneficial for reducing time-space complexity, simplifying distance metric calculation, and hence improving learning efficiency. More discussions about adverse effect of high dimensional space on distance metric calculation can be found in [18].

An autoencoder can be defined as a deep learning algorithm that consists of a symmetrical network, with a certain hidden layer called bottleneck layer. The left half of the network learns an encoder function $\text{enc}(\cdot)$ and generates the output $z_t = \text{sig}(\text{enc}(s_t))$ through the sigmoid activation function $\text{sig}(x) = \frac{1}{1+\exp(-x)}$ to further normalize encoded state vectors. The right half of the network learns a decoder function $\text{dec}(\cdot)$ and generates $\hat{s}_t = \text{dec}(z_t)$. The network is trained by minimizing the loss function $L(s_t, \hat{s}_t)$ that penalizes $\hat{s}_t$ for being dissimilar to s_t in the sense of mean square error. This network training is before initiating the policy optimization process, and it uses the data collected by running a random policy in the relevant environment for certain timesteps, denoted as N_{encode}. It is worth noting that these N_{encode} timesteps are not part of the N timesteps of the policy learning process.

During a policy learning process, the trained encoder network encodes s_t into a compressed state vector z_t, that is stored in the buffer $\mathcal{Z}$ and hence used for state novelty score calculation as described in the next subsection. The buffer updated as each timestep n is explicitly denoted as $\mathcal{Z}_n$, for $n = 1, \cdots, N$.

B. State novelty scorer (SNS)

We first define the density of a state $z^* \in \mathcal{Z}_n$, denoted as $\text{den}(z^*, \mathcal{Z}_n)$. For this purpose, we uniformly sample I mini-batches of size L from the state buffer $\mathcal{Z}_n$. These mini-batches are represented by the sets $P^i = \{z_1^i, z_2^i, \cdots, z_L^i\}$ with $z_l^i \in \mathcal{Z}_n$, for $l = 1, \cdots, L$ and $i = 1, \cdots, I$. It is worth mentioning that mini-batch size is defined as $L = \text{round}(\wp\% \times n)$, where $0 < \wp < 100$ is a constant hyperparameter and the operator $\text{round}(\cdot)$ returns the nearest integer. The density estimation formula is adapted from the density peak clustering algorithm using K nearest neighbors (DPC-KNN) [19]. In particular, we define the approximate density value of z^*, estimated from the mini-batch P^i, as follows:

$$\text{den}_P(z^*, P^i) = e^{-\frac{1}{L}\sum_{l=1}^{L}\{\upsilon_c \|z^* - z_l^i\|\}},\ \upsilon_c = e^{-c\|z^* - z_l^i\|}, \tag{2}$$

where c is a constant and the weight υ_c penalizes the contribution of a datapoint in density calculation based on its distance from the z^*, measured by the Euclidean norm $\|z^* - z_l^i\|$. The approximate density value of z^* can be repeatedly estimated from the I mini-batches and forms a dataset $\{\text{den}_P(z^*, P^1), \cdots, \text{den}_P(z^*, P^I)\}$. It is ready to calculate the density of z^* as the average of this dataset as follows:

$$\text{den}(z^*, \mathcal{Z}_n) = \frac{1}{I}\sum_{i=0}^{I} \text{den}_P(z^*, P^i). \tag{3}$$

The SNS module performs two crucial tasks. Firstly, it calculates the HVD point of the encoded state buffer $\mathcal{Z}_n$, denoted as $\text{HVD}(\mathcal{Z}_n)$. For this purpose, SNS uniformly samples a set of J candidate HVD datapoints from $\mathcal{Z}_n$, denoted as $Q = \{z_1^*, z_2^*, \cdots, z_J^*\}$ with $z_j^* \in \mathcal{Z}_n$, $j = 1, \cdots, J$. It then estimates the density $\text{den}(z_j^*, \mathcal{Z}_n)$ of each candidate datapoint and selects the one with the highest density as the HVD of $\mathcal{Z}_n$, that is,

$$\begin{aligned} &\text{HVD}(\mathcal{Z}_n) \\ &= \begin{cases} \text{argmax}_{z^* \in Q}\{\text{den}(z^*, \mathcal{Z}_n)\}, & n = M, 2M, 3M, \cdots \\ \text{HVD}(\mathcal{Z}_{n-1}), & \text{otherwise} \end{cases}. \end{aligned} \tag{4}$$

It is worth mentioning that HVD is not updated for every timestep, but for every M timesteps. An effective $\mathrm{HVD}(\mathcal{Z}_n)$ can be calculated only for $n \geq M$, that is, the number of samples reaches the cut-in threshold M. With this definition, an HVD point represents the highest density zone of a state butter $\mathcal{Z}_n$, and it also represents clusters of states with high visitation frequency.

Secondly, SNS calculates the novelty score of the current state z_t[1] with respect to the buffer $\mathcal{Z}_n$ according to the Euclidean distance between z_t and the HVD of $\mathcal{Z}_n$, i.e.

$$\eta(z_t, \mathcal{Z}_n) = \|z_t - \mathrm{HVD}(\mathcal{Z}_n)\|. \tag{5}$$

Let us define the absolute density of z^* in $\mathcal{Z}_n$ as

$$\text{abs-den}(z^*, \mathcal{Z}_n) = e^{-\frac{1}{n}\sum_{z\in\mathcal{Z}_n}\{v_c\|z^*-z\|\}},\ v_c = e^{-c\|z^*-z\|},$$

and hence the absolute HVD of $\mathcal{Z}_n$ as the point with the highest absolute density, that is,

$$\text{abs-HVD}(\mathcal{Z}_n) = \mathrm{argmax}_{z^*\in\mathcal{Z}_n}\{\text{abs-den}(z^*, \mathcal{Z}_n)\}.$$

These absolute values can be calculated by exhausting all the points in $\mathcal{Z}_n$ using (2), (3), and (4) with $I = 1$, $L = n$, and $J = n$. However, it becomes infeasible when the size of $\mathcal{Z}_n$ increases. Therefore, in our algorithms, we use only a certain amount of samples to calculate the HVD point, which is practically representative of high density zones.

C. Plausible novelty scorer (PNS)

The PNS module is to measure the so-called plausible novelty (PN) score of the current state z_t with respect to the buffer $\mathcal{Z}_n$, which is a combined score of the state's novelty score and the benefit of exploring it towards policy optimization. The former has been calculated as $\eta(z_t, \mathcal{Z}_n)$ and the latter can be quantified as the V-value of the state, $V(s_t)$, which measures its expected return. It is noted that V-value is calculated based on the original state s_t rather than the encoded state z_t.

V-value is calculated by a V-network, parameterized by θ and denoted as V_θ. The simplest update of the network parameter vector $\theta_{n+1} \leftarrow \theta_n$, according to the sample (s_t, r_t, s_{t+1}), can be

$$V_{\theta_{n+1}}(s_t) \leftarrow V_{\theta_n}(s_t) + \mu\delta_t,$$
$$\delta_t = r_t + \gamma V_{\theta_n}(s_{t+1}) - V_{\theta_n}(s_t),$$

where δ_t is the temporal-difference (TD) error minimized using a gradient descent approach with the network learnt over time; μ is a learning rate. The V-network is trained by using the same samples that train the primary algorithm's policy and critic networks. This choice is for the proposed artifacts to be easily merged with a conventional off-policy AC algorithm without any substantial change to its existing architecture.

As a result, the PN score of the current state (s_t and z_t) with respect to the buffer $\mathcal{Z}_n$, based on the V-network V_{θ_n}, is defined as follows:

$$\xi(s_t, z_t, \mathcal{Z}_n) = \eta(z_t, \mathcal{Z}_n) \times V_{\theta_n}(s_t). \tag{6}$$

[1]More specifically, it should be denoted as z_t^e that represents the current state of the e-th episode. We ignore the superscript e for notation simplicity.

D. Intrinsic reward generator (IRG)

The IRG module receives the PN score $\xi(s_t, z_t, \mathcal{Z}_n)$ of the current state and normalizes it based on its significance in its local neighborhood. This normalized value is regarded as a new intrinsic reward, that is incorporated with the reward r_t to make an augmented reward.

For normalization of a PN score, IRG generates K samples around the current state vector z_t, denoted as

$$\hat{z}_t^k = z_t + \rho,\ \rho \sim \mathcal{N}(0, 0.1),\ k = 1, 2, \cdots, K.$$

Also denote the set of neighboring samples of z_t as $\hat{Z}_t = \{\hat{z}_t^1, \cdots, \hat{z}_t^K\}$. For each sample $\hat{z}_t^k$, the PN score is calculated as $\xi(\mathrm{dec}(\hat{z}_t^k), \hat{z}_t^k, \mathcal{Z}_n)$, where the original version of $\hat{z}_t^k$ is not available but it can be estimated as $\mathrm{dec}(\hat{z}_t^k)$ by the right half of the autoencoder network, i.e., the decoder. Then, the highest PN score of the K samples is defined as

$$\xi_{\max}(z_t, \mathcal{Z}_n) = \max_{\hat{z}_t\in\hat{Z}_t}\{\xi(\mathrm{dec}(\hat{z}_t), \hat{z}_t, \mathcal{Z}_n)\}$$

and hence the intrinsic reward defined as

$$\zeta(s_t, z_t, \mathcal{Z}_n) = \frac{2}{e^{\tilde{\xi}} + e^{-\tilde{\xi}}}$$
$$\tilde{\xi} = \xi_{\max}(z_t, \mathcal{Z}_n) - \xi(s_t, z_t, \mathcal{Z}_n).$$

Here, $\tilde{\xi}$ is the difference between the highest PN score of the neighboring samples of z_t and its own PN score. It is noted that the intrinsic reward $\zeta(s_t, z_t, \mathcal{Z}_n) \in (0, 1]$. When the difference $\tilde{\xi} = 0$, that is, z_t also receives the highest PN score of its neighboring samples, the intrinsic reward $\zeta(s_t, z_t, \mathcal{Z}_n) = 1$ is maximized; otherwise, $\zeta(s_t, z_t, \mathcal{Z}_n) < 1$.

For a given extrinsic reward r_t of the primary algorithm and an intrinsic reward $r_t^{\mathrm{intr}} = \zeta(s_t, z_t, \mathcal{Z}_n)$ as calculated above, we can construct an augmented reward as follows:

$$r_t^{\mathrm{aug}} = (1-\beta)r_t + \beta\zeta(s_t, z_t, \mathcal{Z}_n), \tag{7}$$

where $\beta \in [0, 1)$ is a regularization coefficient that controls the infusion of the intrinsic reward into the extrinsic reward. In some scenarios, an epsilon-greedy method is used to control the probability of assigning an intrinsic reward, that is,

$$r_t^{\mathrm{aug}} = \begin{cases} (1-\beta)r_t + \beta\zeta(s_t, z_t, \mathcal{Z}_n), & \text{probability } 1-\epsilon \\ r_t, & \text{probability } \epsilon \end{cases}$$

for $\epsilon \in [0, 1)$. In the paper, we use $\epsilon = 0$ unless a nonzero value is explicitly specified. It is also worth mentioning that SAC already contains an intrinsic reward proportional to the policy's entropy $\mathcal{H}(\pi_\phi(\cdot|s_t))$. When IPNS is applied on SAC, the augmented reward becomes

$$r_t^{\mathrm{aug}} = (1-\beta)(r_t + \alpha\mathcal{H}(\pi_\phi(\cdot|s_t))) + \beta\zeta(s_t, z_t, \mathcal{Z}_n).$$

The three reward regularization coefficients α, β, ϵ are hyperparameters in a learning process and a grid search can be adopted to determine appropriate values in different environments.

IV. MUJOCO EXPERIMENTS

The discussion in this section is based on the experiments of a MuJoco InvertedDoublePendulum-v2 (InvDP). They were implemented in OpenAI Gym using a computer with a six-core Intel(R) Core(TM) i7-8750H CPU@2.20GHz. The objective is to study performance improvement by application of the IPNS module over three prominent off-policy AC type algorithms: SAC, DDPG, and TD3.

After a policy is trained for every certain steps (called one training unit for convenience), its performance is immediately evaluated by running the corresponding deterministic policy for five consecutive episodes. One training unit is $2,000$ steps and there are $U = 50$ training units (the corresponding total training steps are 0.1 million). The average return over five evaluation episodes is regarded as the episodic return $\mathcal{R}_u^\omega$ for the training unit $u = 1, \cdots, U$. For each algorithm, this process is repeated for five times with $\omega = 1, \cdots, 5$. Each repeated run is with a different random seed. The same set of five random seeds was used for every pair of primary and IPNS algorithms for fair comparison.

The autoencoder models used for different environments were trained using $N_{\text{encode}} = 10,000$ datapoints. These training datapoints or state vectors were obtained by running a random policy in relevant environments. We chose an appropriate dimension of bottleneck layer such that the autoencoder's training loss $L(s_t, \hat{s}_t)$ is as low as with one significant digit (~ 0.01).

The evolution curves of the episodic return versus the number of training units (i.e. in terms of the total number of training steps) are plotted in the figures in this section. A solid curve indicates the mean of the five repeated runs, i.e., $\bar{\mathcal{R}}_u = \sum_{\omega=1}^{5} \mathcal{R}_u^\omega / 5$ and the shaded area shows the confidence interval of the repeats representing the corresponding standard deviation σ_u. Each curve is smoothed using its moving average of eleven training units for clarity of understanding. The efficiency of the IPNS strategy is discussed for InvertedDoublePendulum-v2 including HVD estimation, intrinsic reward calculation, ablation evaluation, and performance comparison with benchmarks.

1) HVD estimation: The HVD point estimation in the SNS module involves three parameters, J, I, and $\wp$. For fast computational purpose, we intent to keep J to a small number but not compromising with the quality of HVD estimation. It is evident from the experiments that a higher value of J, i.e. $J \geq 10$ makes the HVD point be well close to the two peaks. Also, the HVD estimations due to these values are almost similar. Therefore, we choose $J = 10$ for the current environment. Using the similar argument, we choose $I = 100$ and $\wp = 1$.

The calculation of HVD is updated for every M timesteps. Training performance in the experiments with different values of M is shown in Fig. 2. When M is relatively small, e.g., $M = 200$, an HVD point is updated too frequently. The estimates are significantly affected by transient density distribution variations especially when the buffer is comparatively empty, which results in fluctuation in estimated values

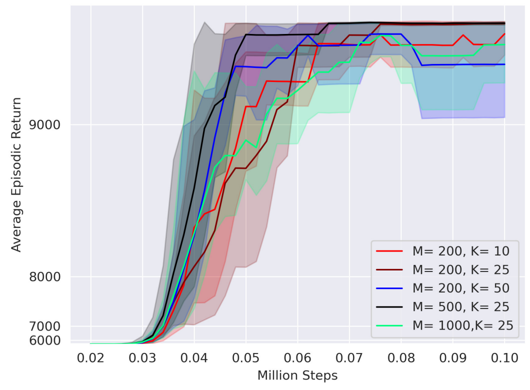

Fig. 2: Training performance of SAC+IPNS with different update step size M and different sample number K.

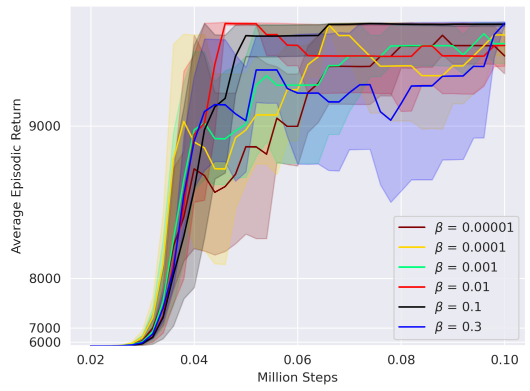

Fig. 3: Training performance SAC+IPNS with different intrinsic reward regularization coefficient β.

over time and hence leads to slow policy learning. For a too large M, e.g., $M = 1000$, an HVD point likely becomes outdated and poorly represents the latest data distribution, which may score a state's novelty erroneously. Based on these observations, we select $M = 500$ to be comparatively better for the environment.

2) Intrinsic reward calculation: IRG generates K samples around the current state vector to calculate its intrinsic reward. We also run experiments to study the effect of K on training performance. It is observed in Fig. 2 that a small $K = 10$ leads to poor normalization of PNS score, resulting in learning an inferior policy; a policy with a large $K = 50$ acts myopically and shows initial rise, but fails to maintain its performance with time. We choose $K = 25$ that enables good normalization and hence superior performance.

Another important parameter for intrinsic reward assignment is the intrinsic reward regularization coefficient β. The influence of β on training performance is demonstrated in Fig. 3. We select $\beta = 0.1$ with which the best training performance is achieved by SAC+IPNS.

3) Ablation evaluation of IPNS: Experiments using different modules of IPNS were conducted to study their unique contributions, as shown in Fig. 4. We start the evaluation with simple SAC+ SNS and SAC+ PNS without SE. It is observed that SAC+ SNS picks up performance earlier

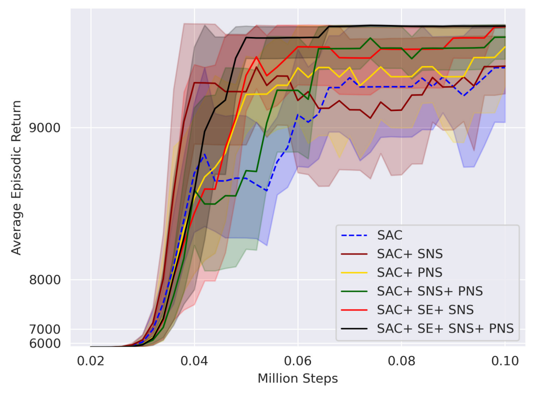

Fig. 4: Ablation evaluation of individual artifacts of the IPNS algorithm.

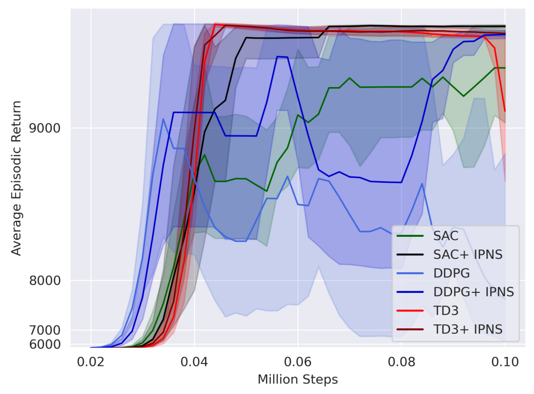

Fig. 5: Performance comparison of IPNS with other benchmarks for InvertedDoublePendulum-v2.

than SAC but it fails to generalize well with time; SAC+ PNS gives a relatively poor performance since without SNS the PN score ξ is not weighted by the novelty score. The evaluation shows the necessity of using both modules, i.e., SAC+ SNS+ PNS, which provides better generalization over time compared to SAC+ SNS, although it takes a long time to learn superior performance. Next, we examine the function of the SE module. The improvement achieved by SAC+SE+SNS over SAC+SNS shows the efficacy of encoding states using the SE module as it gives a condensed, normalized latent representation of states and hence leads to faster policy learning and improved generalization over time. Finally, adding SE to SAC+ SNS+ PNS forms a complete SAC+IPNS, which demonstrates best performance utilizing all the proposed artifacts.

4) Comparison with benchmarks: The comparison is graphically exhibited in Fig. 5. The comparison shows that IPNS performs consistently well when paired with any of the three conventional off-policy algorithms.

V. CONCLUSION

We have proposed a new IPNS strategy for accelerating exploration in off-policy AC algorithms and thereby improving sample efficiency. The key idea is to incentivize exploration towards the states of high plausible novelty scores through a properly designed intrinsic reward. Plausible novelty of a state consists of both state novelty and the chance of positively impacting policy optimization by visiting the state. An interesting feature of IPNS is its easy implementation by integrating it with any primary off-policy AC algorithm without major modification. Three state-of-art off-policy AC algorithms have been tested as the primary algorithms to verify the substantial improvement in learning performance by IPNS, in terms of sample efficiency, stability, and performance variance.

REFERENCES

[1] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” *Advances in Neural Information Processing Systems*, pp. 1008–1014, 2000.
[2] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” *International Conference on Machine Learning*, pp. 387–395, 2014.
[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
[4] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” *International Conference on Machine Learning*, pp. 1587–1596, 2018.
[5] C. Banerjee, Z. Chen, N. Noman, and M. Zamani, “Optimal actor-critic policy with optimized training datasets,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2022.
[6] T. Rückstiess, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber, “Exploring parameter space in reinforcement learning,” *Paladyn*, vol. 1, no. 1, pp. 14–24, 2010.
[7] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying count-based exploration and intrinsic motivation,” *Advances in Neural Information Processing Systems*, vol. 29, 2016.
[8] R. Zhao and V. Tresp, “Curiosity-driven experience prioritization via density estimation,” *arXiv preprint arXiv:1902.08039*, 2019.
[9] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, “# exploration: A study of count-based exploration for deep reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[10] B. C. Stadie, S. Levine, and P. Abbeel, “Incentivizing exploration in reinforcement learning with deep predictive models,” *arXiv preprint arXiv:1507.00814*, 2015.
[11] D. Pathak, D. Gandhi, and A. Gupta, “Self-supervised exploration via disagreement,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 5062–5071.
[12] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning with deep energy-based policies,” *International Conference on Machine Learning*, pp. 1352–1361, 2017.
[13] C. Zhang, Y. Cai, L. Huang, and J. Li, “Exploration by maximizing Rényi entropy for reward-free RL framework,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 859–10 867.
[14] M. Yuan, M.-o. Pun, and D. Wang, “Rényi state entropy for exploration acceleration in reinforcement learning,” *arXiv preprint arXiv:2203.04297*, 2022.
[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *International Conference on Machine Learning*, pp. 1861–1870, 2018.
[16] C. Banerjee, Z. Chen, and N. Noman, “Improved soft actor-critic: Mixing prioritized off-policy samples with on-policy experiences,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022, DOI: 10.1109/TNNLS.2022.3174051.
[17] J. Jiang, M. Fu, and Z. Chen, “Physics informed intrinsic rewards in reinforcement learning,” *Proceedings of 2022 Australian and New Zealand Control Conference*, 2022.
[18] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, “On the surprising behavior of distance metrics in high dimensional space,” in *International conference on database theory*. Springer, 2001, pp. 420–434.
[19] M. Du, S. Ding, and H. Jia, “Study on density peaks clustering based on K-nearest neighbors and principal component analysis,” *Knowledge-Based Systems*, vol. 99, pp. 135–145, 2016.