STATE ENTROPY MAXIMIZATION WITH RANDOM ENCODERS FOR EFFICIENT EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent exploration methods have proven to be a recipe for improving sample-efficiency in deep reinforcement learning (RL). However, efficient exploration in high-dimensional observation spaces still remains a challenge. This paper presents Random Encoders for Efficient Exploration (RE3), an exploration method that utilizes state entropy as an intrinsic reward. In order to estimate state entropy in environments with high-dimensional observations, we utilize a k-nearest neighbor entropy estimator in the low-dimensional representation space of a convolutional encoder. In particular, we find that the state entropy can be estimated in a stable and compute-efficient manner by utilizing a randomly initialized encoder, which is fixed throughout training. Our experiments show that RE3 significantly improves the sample-efficiency of both model-free and model-based RL methods on locomotion and navigation tasks from DeepMind Control Suite and MiniGrid benchmarks. We also show that RE3 allows learning diverse behaviors without extrinsic rewards, effectively improving sample-efficiency in downstream tasks.

1 INTRODUCTION

Exploration remains one of the main challenges of deep reinforcement learning (RL) in complex environments with high-dimensional observations. Many prior approaches to incentivizing exploration introduce intrinsic rewards based on a measure of state novelty. These include count-based visitation bonuses (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017) and prediction errors (Stadie et al., 2015; Houthooft et al., 2016; Burda et al., 2019; Pathak et al., 2017). By introducing such novelty-based intrinsic rewards, these approaches encourage agents to visit diverse states, but leave unanswered the fundamental question of how to quantify effective exploration in a principled way.

To address this limitation, Lee et al. (2019) and Hazan et al. (2019) proposed that exploration methods should encourage uniform (i.e., maximum entropy) coverage of the state space. For practical state entropy estimation without learning density models, Mutti et al. (2020) estimate state entropy by measuring distances between states and their k-nearest neighbors. To extend this approach to high-dimensional environments, recent works (Badia et al., 2020; Tao et al., 2020; Liu & Abbeel, 2021) have proposed to utilize the k-nearest neighbor state entropy estimator in a low-dimensional latent representation space. The latent representations are learned by auxiliary tasks such as dynamics learning (Tao et al., 2020), inverse dynamics prediction (Badia et al., 2020), and contrastive learning (Liu & Abbeel, 2021). However, these methods still involve optimizing multiple objectives throughout RL training. Given the added complexity (e.g., hyperparameter tuning), instability, and computational overhead of optimizing auxiliary losses, it is important to ask whether effective state entropy estimation is possible without introducing additional learning procedures.

In this paper, we present RE3: **R**andom **E**ncoders for **E**fficient **E**xploration, a simple, compute-efficient method for exploration without introducing additional models or representation learning. The key idea of RE3 is to utilize a k-nearest neighbor state entropy estimator in the representation space of a randomly initialized encoder, which is fixed throughout training. Our main hypothesis is that a randomly initialized encoder can provide a meaningful representation space for state entropy estimation by exploiting the strong prior of convolutional architectures. Ulyanov et al. (2018) and Caron et al. (2018) found that the structure alone of deep convolutional networks is a powerful inductive bias that allows relevant features to be extracted for tasks such as image generation and

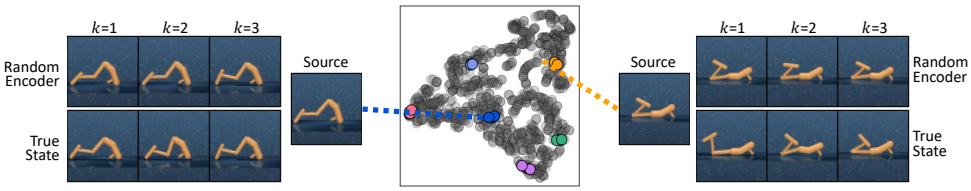


Figure 1: Visualization of k-nearest neighbors of states found by measuring distances in the representation space of a randomly initialized encoder (Random Encoder) and ground-truth state space (True State) on the Hopper environment from DeepMind Control Suite (Tassa et al., 2020). We observe that the representation space of a random encoder effectively captures information about the similarity between states without any representation learning.

classification. In our case, we find that the representation space of a randomly initialized encoder effectively captures information about similarity between states, as shown in Figure 1. Based upon this observation, we propose to maximize a state entropy estimate in the fixed representation space of a randomly initialized encoder.

We highlight the main contributions of this paper below:

- RE3 significantly improves the sample-efficiency of both model-free and model-based RL methods on widely used DeepMind Control Suite (Tassa et al., 2020) and MiniGrid (Chevalier-Boisvert et al., 2018) benchmarks.
- RE3 encourages exploration without introducing representation learning or additional models, outperforming state entropy maximization schemes that involve representation learning and exploration methods that introduce additional models for exploration (Burda et al., 2019; Pathak et al., 2017).
- RE3 is compute-efficient as it does not require gradient computations and updates for additional representation learning, making it a scalable and practical approach to exploration.
- RE3 allows learning diverse behaviors in environments without extrinsic rewards; we further improve sample-efficiency in downstream tasks by fine-tuning a policy pre-trained with the RE3 objective.

2 Related Work

Exploration in reinforcement learning. Exploration algorithms encourage the RL agent to visit a wide range of states by injecting noise to the action space (Lillicrap et al., 2015) or parameter space (Fortunato et al., 2018; Plappert et al., 2018), maximizing the entropy of the action space (Ziebart, 2010; Haarnoja et al., 2018), and setting diverse goals that guide exploration (Florensa et al., 2018; Nair et al., 2018; Colas et al., 2019; Pong et al., 2020). Another line of exploration algorithms introduce intrinsic rewards proportional to prediction errors (Houthooft et al., 2016; Pathak et al., 2017; Burda et al., 2019), and count-based state novelty (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017). Our approach differs in that we explicitly encourage the agent to uniformly visit all states by maximizing the entropy of the state distribution, instead of depending on metrics from additional models.

State entropy maximization. Most closely related to our work are methods that maximize the entropy of state distributions. Hazan et al. (2019) and Lee et al. (2019) proposed to maximize state entropy estimated by approximating the state density distribution. Instead of approximating complex distributions, Mutti et al. (2020) proposed to maximize a k-nearest neighbor state entropy estimate from on-policy transitions. Recent works extend this method to environments with high-dimensional observations. Tao et al. (2020) employ model-based RL techniques to build a representation space for the state entropy estimate that measures similarity in dynamics, and Badia et al. (2020) proposed to measure similarity in the representation space learned by inverse dynamics prediction. The work closest to ours is Liu & Abbeel (2021), which uses off-policy RL algorithms to maximize the k-nearest neighbor state entropy estimate in contrastive representation space (Srinivas et al., 2020) for unsupervised pre-training. We instead explore the idea of utilizing a fixed random encoder to obtain a stable entropy estimate without any representation learning.

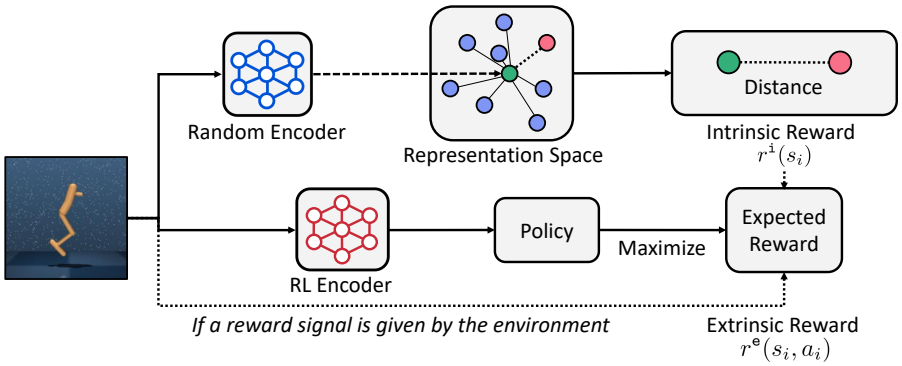


Figure 2: Illustration of RE3. The intrinsic reward for each observation is computed as the distance to its k-nearest neighbor, measured between low-dimensional representations obtained from the fixed random encoder. The intrinsic reward is combined with extrinsic reward from the environment, if present. A separate RL encoder is introduced for a policy that maximizes expected reward.

Random encoders. Random weights have been utilized in neural networks since their beginnings, most notably in a randomly initialized first layer (Gamba et al., 1961) termed the Gamba perceptron by Minsky & Papert (1969). Moreover, nice properties of random projections are commonly exploited for low-rank approximation (Vempala, 2005; Rahimi & Recht, 2007). These ideas have since been extended to deep convolutional networks, where random weights are surprisingly effective at image generation and restoration (Ulyanov et al., 2018), image classification and detection (Caron et al., 2018), and fast architecture search (Saxe et al., 2011). In natural language processing, Wieting & Kiela (2019) demonstrated that learned sentence embeddings show marginal performance gain over random embeddings. In the context of RL, Gaier & Ha (2019) showed that competitive performance can be achieved by architecture search over random weights without updating weights, and Lee et al. (2020) utilized randomized convolutional neural networks to improve the generalization of deep RL agents. Building on these works, we show that random encoders can also be useful for efficient exploration in environments with high-dimensional observations.

3 Method

3.1 Preliminaries

We formulate a control task with high-dimensional observations as a partially observable Markov decision process (POMDP; Kaelbling et al. 1998; Sutton & Barto 2018), which is defined as a tuple $(\mathcal{O}, \mathcal{A}, p, r^{\mathsf{e}}, \gamma)$. Here, $\mathcal{O}$ is the high-dimensional observation space, $\mathcal{A}$ is the action space, $p\left(o'|o_{\leq t}, a_t\right)$ is the transition dynamics, $r^{\mathsf{e}} : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ is the reward function that maps the current observation and action to a reward $r^{\mathsf{e}}_t = r^{\mathsf{e}}\left(o_{\leq t}, a_t\right)$, and $\gamma \in [0, 1)$ is the discount factor. By following common practice (Mnih et al., 2015), we reformulate the POMDP as an MDP (Sutton & Barto, 2018) by stacking consecutive observations into a state $s_t = \{o_t, o_{t-1}, o_{t-2}, ...\}$. For simplicity of notation, we redefine the reward function as $r^{\mathsf{e}}_t = r^{\mathsf{e}}\left(s_t, a_t\right)$. The goal of RL is to learn a policy $\pi(a_t|s_t)$ that maximizes the expected return defined as the total accumulated reward.

***k*-nearest neighbor entropy estimator.** Let X be a random variable with a probability density function p whose support is a set $\mathcal{X} \subset \mathbb{R}^q$. Then its differential entropy is given as $\mathcal{H}(X) = -\mathbb{E}_{x\sim p(x)}[\log p(x)]$. When the distribution p is not available, this quantity can be estimated given N i.i.d realizations of $\{x_i\}_{i=1}^N$ (Beirlant et al., 1997). However, since it is difficult to estimate p with high-dimensional data, particle-based k-nearest neighbors (k-NN) entropy estimator (Singh et al., 2003) can be employed:

$$\widehat{\mathcal{H}}^k_N(X) = \frac{1}{N}\sum_{i=1}^{N} \log \frac{N \cdot ||x_i - x_i^{k\text{-NN}}||_2^q \cdot \widehat{\pi}^{\frac{q}{2}}}{k \cdot \Gamma(\frac{q}{2}+1)} + C_k \tag{1}$$

$$\propto \frac{1}{N}\sum_{i=1}^{N} \log ||x_i - x_i^{k\text{-NN}}||_2, \tag{2}$$

where $x_i^{k\text{-NN}}$ is the k-NN of x_i within a set $\{x_i\}_{i=1}^N$, $C_k = \log k - \Psi(k)$ a bias correction term, Ψ the digamma function, Γ the gamma function, q the dimension of x, $\widehat{\pi} \approx 3.14159$, and the transition from (1) to (2) always holds for $q > 0$.

Algorithm 1 RE3: Off-policy RL version

Initialize parameters of random encoder θ, policy ϕ, replay buffer $\mathcal{B} \leftarrow \emptyset$
for each timestep t **do**
  Collect a transition $\tau_t = (s_t, a_t, s_{t+1}, r_t^{\mathtt{e}})$ using policy π_ϕ // COLLECT TRANSITIONS
  Get a fixed representation $y_t = f_\theta(s_t)$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\tau_t, y_t)\}$
  Sample random minibatch $\{(\tau_j, y_j)\}_{j=1}^{B} \sim \mathcal{B}$ // COMPUTE INTRINSIC REWARD
  for $j = 1$ **to** B **do**
    Compute the distance $||y_j - y||_2$ for all $y \in \mathcal{B}$ and find the k-nearest neighbor $y_j^{k\text{-NN}}$
    Compute $r_j^{\mathtt{i}} \leftarrow \log(||y_j - y_j^{k\text{-NN}}||_2 + 1)$
    Update $\beta_t \leftarrow \beta_0(1-\rho)^t$
    Let $r_j^{\mathtt{total}} \leftarrow r_j^{\mathtt{e}} + \beta_t \cdot r_j^{\mathtt{i}}$
  end for
  Update ϕ with transitions $\{(s_j, a_j, s_{j+1}, r_j^{\mathtt{total}})\}_{j=1}^{B}$ // UPDATE POLICY
end for

3.2 RANDOM ENCODERS FOR EFFICIENT EXPLORATION

We present Random Encoders for Efficient Exploration (RE3), which encourages exploration in high-dimensional observation spaces by maximizing state entropy. The key idea of RE3 is k-nearest neighbor entropy estimation in the low-dimensional representation space of a randomly initialized encoder. To this end, we propose to compute the distance between states in the representation space of a random encoder f_θ whose parameters θ are randomly initialized and fixed throughout training. The main motivation arises from our observation that distances in the representation space of f_θ are already useful for finding similar states without any representation learning (see Figure 1).

State entropy estimate as intrinsic reward. To define the intrinsic reward proportional to state entropy estimate by utilizing (2), we follow the idea of Liu & Abbeel (2021) that treats each transition as a particle, hence our intrinsic reward is given as follows:

$$r^{\mathtt{i}}(s_i) := \log(||y_i - y_i^{k\text{-NN}}||_2 + 1), \tag{3}$$

where $y_i = f_\theta(s_i)$ is a fixed representation from a random encoder and $y_i^{k\text{-NN}}$ is the k-nearest neighbor of y_i within a set of N representations $\{y_1, y_2, ..., y_N\}$. Our intuition is that measuring distance between states in the fixed representation space produces a more stable intrinsic reward as the distance between a given pair of states does not change during training. To compute distances in latent space in a compute-efficient manner, we propose to additionally store low-dimensional representations y in the replay buffer $\mathcal{B}$ during environment interactions. Therefore, we avoid processing high-dimensional states through an encoder for obtaining representations at every RL update. Moreover, we can feasibly compute the distance of y_i to all entries $y \in \mathcal{B}$, in contrast to existing approaches that utilize on-policy samples (Mutti et al., 2020), or samples from a minibatch (Liu & Abbeel, 2021). Our scheme enables stable, precise entropy estimation in a compute-efficient manner.

The RE3 objective. We propose to utilize the intrinsic reward $r^{\mathtt{i}}$ for (a) *online RL*, where the agent solves target tasks guided by extrinsic reward $r^{\mathtt{e}}$ from environments, and (b) *unsupervised pre-training*, where the agent learns to explore the high-dimensional observation space of environments in the absense of extrinsic rewards, i.e., $r^{\mathtt{e}} = 0$. This exploratory policy from pre-training, in turn, can be used to improve the sample-efficiency of the agent in downstream tasks by fine-tuning. Formally, we introduce a policy π_ϕ, parameterized by ϕ, that maximizes the expected return $\mathbb{E}_{\pi_\phi}\left[\sum_{j=0}^{\infty} \gamma^j r_j^{\mathtt{total}}\right]$, where the total reward $r_j^{\mathtt{total}}$ is defined as:

$$r_j^{\mathtt{total}} := r^{\mathtt{e}}(s_j, a_j) + \beta_t \cdot r^{\mathtt{i}}(s_j), \tag{4}$$

where $\beta_t \geq 0$ is a hyperparameter that determines the tradeoff between exploration and exploitation at training timestep t. We use the exponential decay schedule for β_t throughout training to encourage the agent to further focus on extrinsic reward from environments as training proceeds, i.e., $\beta_t = \beta_0(1-\rho)^t$, where ρ is a decay rate. While the proposed intrinsic reward would converge to 0 as more similar states are collected during training, we discover that decaying β_t empirically stabilizes the performance. We provide the full procedure for RE3 with off-policy RL in Algorithm 1 and RE3 with on-policy RL in Algorithm 2.

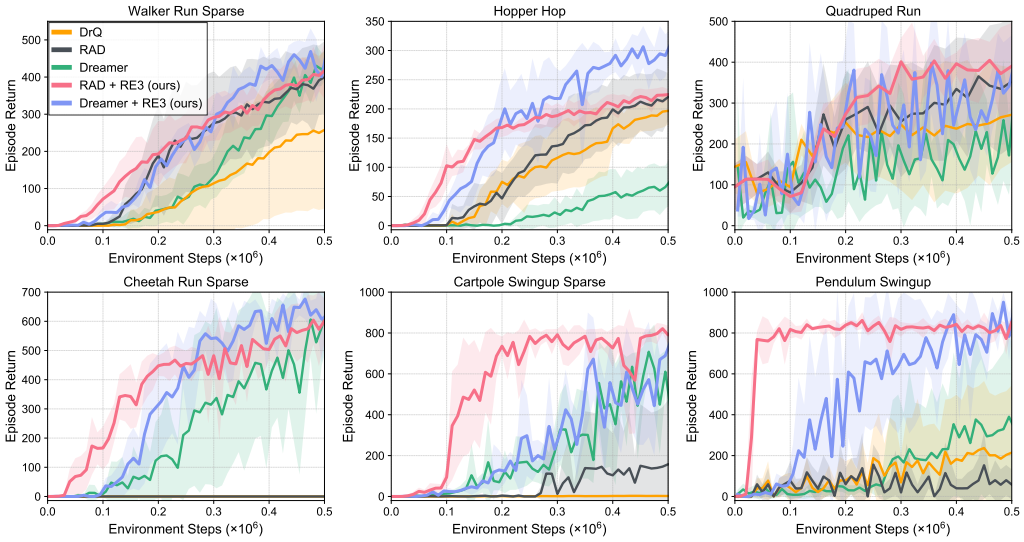


Figure 3: Performance on locomotion tasks from DeepMind Control Suite. RE3 consistently improves the sample-efficiency of RAD and Dreamer. The solid line and shaded regions represent the mean and standard deviation, respectively, across five runs.

4 Experiments

We designed experiments to answer the following questions:

- Can RE3 improve the sample-efficiency of both model-free and model-based RL algorithms (see Figure 3)?
- How does RE3 compare to state entropy maximization schemes that involve representation learning (see Figure 4) and other exploration schemes that introduce additional models for exploration (see Figure 5)?
- Can RE3 further improve the sample-efficiency of off-policy RL algorithms by unsupervised pre-training (see Figure 6 and Figure 7a)?
- How compute-efficient is RE3 (see Figure 7b)?
- Can RE3 improve the sample-efficiency of on-policy RL in discrete control (see Figure 8)?

4.1 DeepMind Control Suite Experiments

Setup. To evaluate the sample-efficiency of our method, we compare to Dreamer (Hafner et al., 2020), a state-of-the-art model-based RL method for visual control; and two state-of-the-art model-free RL methods, RAD (Laskin et al., 2020) and DrQ (Kostrikov et al., 2021). For comparison with other exploration methods, we consider RND (Burda et al., 2019) and ICM (Pathak et al., 2017) that introduce additional models for exploration. For RE3 and baseline exploration methods, we use RAD as the underlying model-free RL algorithm. To further demonstrate the applicability of RE3 to model-based RL algorithms, we also consider a combination of Dreamer and RE3. For random encoders, we use convolutional neural networks with the same architecture as underlying RL algorithms, but with randomly initialized parameters fixed during the training. As for the newly introduced hyperparameters, we use $k = 3$, $\beta_0 = 0.05$, and $\rho \in \{0.0, 0.00001, 0.000025\}$. We provide more details in Appendix B and source code in Appendix A.

Comparative evaluation. Figure 3 shows that RE3 consistently improves the sample-efficiency of RAD on various tasks. In particular, RAD + RE3 achieves average episode return of 601.6 on Cheetah Run Sparse, where both model-free RL methods RAD and DrQ fail to solve the task. We emphasize that state entropy maximization with RE3 achieves such sample-efficiency with minimal cost due to its simplicity and compute-efficiency. We also observe that Dreamer + RE3 improves the sample-efficiency of Dreamer on most tasks, which demonstrates the applicability of RE3 to both model-free and model-based RL. We provide experimental results on more tasks in Figure 9.

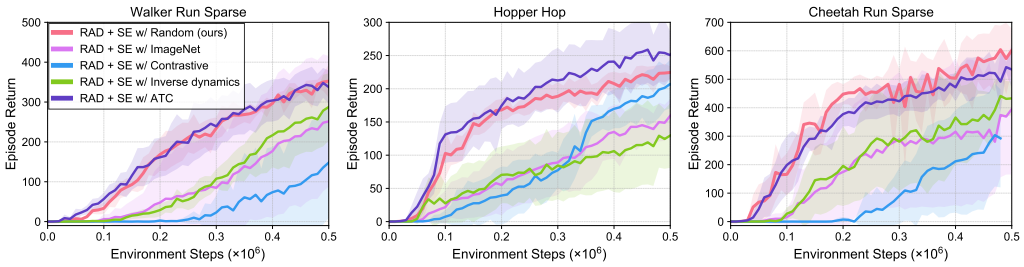


Figure 4: We compare state entropy (SE) maximization with RE3 to state entropy maximization schemes that involve representation learning. The solid line and shaded regions represent the mean and standard deviation, respectively, across five runs.

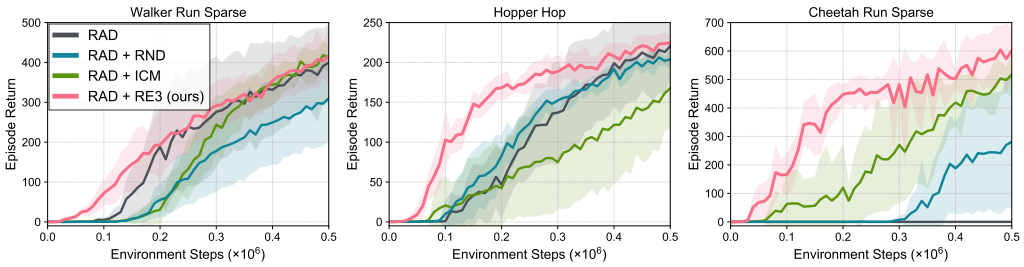


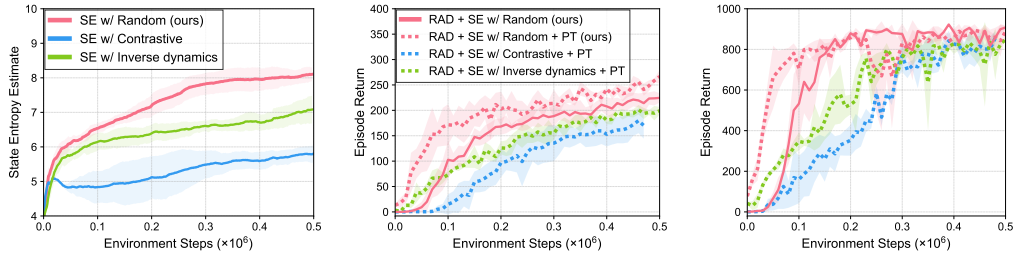
Figure 5: Performance on locomotion tasks from DeepMind Control Suite. RAD + RE3 outperforms other exploration methods in terms of sample-efficiency. The solid line and shaded regions represent the mean and standard deviation, respectively, across five runs.

Effects of representation learning. To better grasp how RE3 improves sample-efficiency, we compare to state entropy maximization schemes that involve representation learning in Figure 4. Specifically, we consider a convolutional encoder trained by contrastive learning (RAD + SE w/ Contrastive) and inverse dynamics prediction (RAD + SE w/ Inverse dynamics). We also consider two pre-trained encoders whose parameters are fixed throughout training: a ResNet-50 (He et al., 2016) encoder pre-trained on ImageNet dataset (RAD + SE w/ ImageNet) and an ATC (Stooke et al., 2020) encoder pre-trained by contrastive learning on pre-training datasets that contain a wide range of states from the environments (RAD + SE w/ ATC), where more details are available at Appendix B. We found that our method (RAD + SE w/ Random) exhibits better sample-efficiency than approaches that continually update representations throughout training (RAD + SE w/ Contrastive, RAD + SE w/ Inverse dynamics). This demonstrates that utilizing fixed representations helps improve sample-efficiency by enabling stable state entropy estimation throughout training. We also observe that our approach outperforms RAD + SE w/ ImageNet, implying that it is not necessarily beneficial to employ a pre-trained encoder, and fixed random encoders can be effective for state entropy estimation without having been trained on any data. We remark that representations from the pre-trained ImageNet encoder could not be useful for our setup, due to the different visual characteristics of natural images in the ImageNet dataset and image observations in our experiments. However, our approach also achieves comparable sample-efficiency to RAD + SE w/ ATC, which shows that random encoders can provide a stable state entropy estimation even competitive with encoders pre-trained on data from the environments.

Comparison with other exploration methods. We also compare our state entropy maximization scheme to other exploration methods combined with RAD, i.e., RAD + RND and RAD + ICM, that learn additional models to obtain intrinsic rewards proportional to prediction errors. As shown in Figure 5, RAD + RE3 consistently exhibits superior sample efficiency in most tasks (see Figure 10 for results on more tasks). While RND similarly employs a fixed random network for the intrinsic reward, it also introduces an additional network which requires training and therefore suffers from instability.[1] This result demonstrates that RE3 can improve sample-efficiency without introducing additional models for exploration, by utilizing fixed representations from a random encoder for stable state entropy estimation.

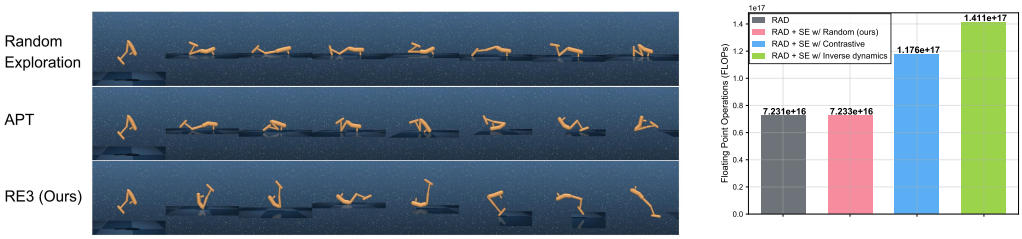
Evaluation of unsupervised pre-training. To evaluate the effectiveness of RE3 for learning diverse behaviors in the pre-training phase without extrinsic rewards, we first visualize the behaviors of policies pre-trained for 500K environment steps in Figure 7a. One can see that the pre-trained policy using RE3 exhibits more diverse behaviors compared to random exploration or APT (Liu

[1] Taïga et al. (2020) also observed that additional techniques were critical to the performance of RND.



(a) Ground-truth entropy estimate (b) Hopper Hop (c) Hopper Stand

Figure 6: (a) We observe that pre-training Hopper agent with RE3 results in a higher state entropy estimate in the ground-truth state space, i.e., proprioceptive state space, compared to other state entropy maximization schemes that involve representation learning. This improves sample-efficiency when fine-tuning pre-trained policies on (b) Hopper Hop and (c) Hopper Stand. The solid (or dotted) line and shaded regions represent the mean and standard deviation, respectively, across three runs.



(a) Behavior visualization (b) Compute-efficiency

Figure 7: (a) Observations from evenly-spaced intervals for one episode of executing policy actions. As a baseline, we show random exploration, i.e. sampling from the action space uniformly at random. We compare the diversity of visited states resulting from pre-training for 500K steps with APT (Liu & Abbeel, 2021) and RE3 (ours). Videos are available in Appendix A. (b) Number of FLOPs used by each agent to achieve its performance at 500K environment steps in Hopper Hop.

& Abbeel, 2021), where a policy is pre-trained to maximize state entropy estimate in contrastive representation space. To further evaluate the diversity of behaviors quantitatively, we also show the state entropy estimate in the ground-truth (i.e., proprioceptive) state space in the Hopper environment in Figure 6a. We observe that RE3 exhibits a higher ground-truth state entropy estimate than state entropy maximization schemes that use contrastive learning and inverse dynamics prediction during pre-training. This implies that RE3 can effectively maximize the ground-truth state entropy without being able to directly observe underlying ground-truth states.

Fine-tuning in downstream tasks. We also remark that the diversity of behaviors leads to superior sample-efficiency when fine-tuning a pre-trained policy in downstream tasks, as shown in Figure 6b and 6c. Specifically, we fine-tune a pre-trained policy in downstream tasks where extrinsic rewards are available, by initializing the parameters of policies with parameters of pre-trained policies (see Appendix B for more details). We found that fine-tuning a policy pre-trained with RE3 (RAD + SE w/ Random + PT) further improves the sample-efficiency of RAD + SE w/ Random, and also outperforms other pre-training schemes. We emphasize that RE3 allows learning such diverse behaviors by pre-training a policy only for 500K environment steps, while previous work (Liu & Abbeel, 2021) reported results by training for 5M environment steps.

Compute-efficiency. We show that RE3 is a practical and scalable approach for exploration in RL due to its compute-efficiency. In particular, RE3 is compute-efficient in that (a) there are no gradient updates through the random encoder, and (b) there are no unnecessary forward passes for obtaining representations at every update step since we store low-dimensional latent representations in the replay buffer. To evaluate compute-efficiency, we show the floating point operations (FLOPs) consumed by RAD, RAD + SE w/ Random (ours), RAD + SE w/ Contrastive, and RAD + SE w/ Inverse dynamics. We account only for forward and backward passes through neural network layers. We explain our full procedure for counting FLOPs in Appendix E. Figure 7b shows the FLOPS used by each agent to achieve its final performance in Hopper Hop (see Figure 4 for corresponding learning curves). Estimating state entropy with a random encoder is significantly more compute-efficient than with a encoder trained with representation learning. One important detail here is that RAD + SE w/ Random has comparable compute-efficiency to RAD. Therefore, we improve the sample-efficiency of RAD without sacrificing compute-efficiency.

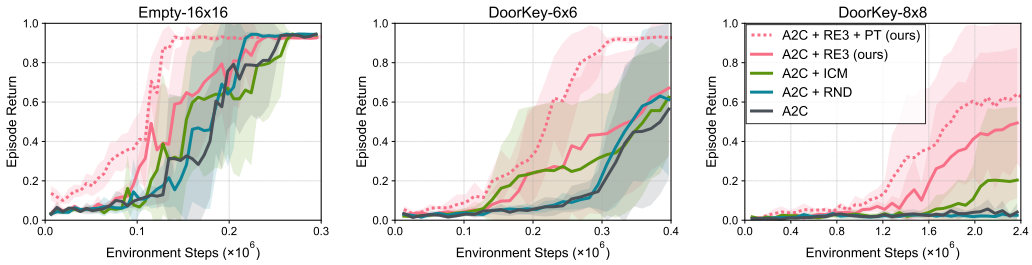


Figure 8: Performance on navigation tasks from MiniGrid. The solid (or dotted) line and shaded regions represent the mean and standard deviation, respectively, across five runs.

4.2 MiniGrid Experiments

Setup. We evaluate our method on MiniGrid (Chevalier-Boisvert et al., 2018), a gridworld environment with a selection of sparse reward tasks. The tasks we consider are shown in Figure 11. For evaluation, we use Advantage Actor-Critic (A2C; Mnih et al. 2016) as the underlying RL algorithm, and consider two exploration methods, RND and ICM. The agent has access to a compact $7 \times 7 \times 3$ embedding of the 7×7 grid directly in front of it, making the environment partially-observable. To combine RE3 with A2C, an on-policy RL method, we maintain a replay buffer of 10K samples solely for computing the RE3 intrinsic reward, and compute k-NN distances between the on-policy batch and the entire replay buffer. For RND and ICM, the intrinsic reward is computed using the on-policy batch. For RE3, ICM, and RND, we perform hyperparameter search over the intrinsic reward weight and report the best result (see Appendix D for more details).

Comparison with other exploration methods. Figure 8 shows that RE3 is more effective for improving the sample-efficiency of A2C in most tasks, compared to other exploration methods, RND and ICM, that learn additional models. In particular, A2C + RE3 achieves average episode return of 0.49 at 2.4M environment steps in DoorKey-8x8; in comparison, A2C + ICM achieves a return of 0.20 and A2C + RND and A2C both fail to achieve non-trivial returns. These results demonstrate that state entropy maximization with RE3 can also improve the sample-efficiency of on-policy RL algorithms by introducing only a small-size replay buffer.

Fine-tuning in downstream tasks. To evaluate the effectiveness of RE3 for unsupervised pre-training in MiniGrid tasks, we first pre-train a policy in a large vacant room (Empty-16$\times$16) to maximize RE3 intrinsic rewards for 100K environments steps. Then, we fine-tune the pre-trained policy in downstream tasks by initializing a policy with pre-trained parameters and subsequently training with A2C + RE3. Figure 8 shows that A2C + RE3 + PT significantly improves the sample-efficiency of A2C, which demonstrates that the ability to explore novel states in a large empty room helps improve sample-efficiency in DoorKey tasks, which involve the added complexity of additional components, e.g., walls, doors, and locks. We show a comparison to state entropy maximization with contrastive learning in Figure 12 and observe that it does not work well, as contrastive learning depends on data augmentation specific to images (e.g., random shift, random crop, and color jitter), which are not compatible with the compact embeddings used as inputs for MiniGrid. We remark that RE3 eliminates the need for carefully chosen data augmentations by employing a random encoder.

5 Discussion

In this paper, we present RE3, a simple exploration method compatible with both model-free and model-based RL algorithms. RE3 maximizes a k-nearest neighbor state entropy estimate in the fixed representation space of a randomly initialized encoder. Our experimental results demonstrate that RE3 can encourage exploration in widely-used benchmarks, as it enables stable and compute-efficient state entropy estimation. Here, we emphasize that our goal is not to claim that representation learning or additional models are not required for exploration, but to show that fixed random encoders can be useful for efficient exploration. For more visually complex domains, utilizing pre-trained fixed representations for stable state entropy estimation could be more useful, but we leave it to future work to explore this direction further because this would require having access to environments and a wide distribution of states for pre-training, which is itself a non-trivial problem. Another interesting direction would be to investigate the effect of network architectures for state entropy estimation, or to utilize state entropy for explicitly guiding the action of a policy to visit diverse states. We believe RE3 would facilitate future research by providing a simple-to-implement, stable, and compute-efficient module that can be easily combined with other techniques.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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. In *International Conference on Learning Representations*, 2020.

Jan Beirlant, Edward J Dudewicz, László Györfi, and Edward C Van der Meulen. Nonparametric entropy estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39, 1997.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision*, 2018.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International Conference on Machine Learning*, 2019.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, 2018.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. In *International Conference on Learning Representations*, 2018.

Adam Gaier and David Ha. Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Alberto Gamba, L Gamberini, G Palmieri, and R Sanna. Further experiments with papa. *Il Nuovo Cimento (1955-1965)*, 20(2):112–115, 1961.

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*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, 2016.

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *IEEE conference on computer vision and pattern recognition*, 2018.

Jongheon Jeong and Jinwoo Shin. Training cnns with selective allocation of channels. In *International Conference on Machine Learning*, 2019.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, 2020.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

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. In *arXiv preprint arXiv:1509.02971*, 2015.

Hao Liu and Pieter Abbeel. Unsupervised active pre-training for reinforcement learning, 2021. URL https://openreview.net/forum?id=cvNYovr16SB.

Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 1969.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. A policy gradient method for task-agnostic exploration. *arXiv preprint arXiv:2007.04640*, 2020.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, 2018.

Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning*, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, 2017.

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. In *International Conference on Learning Representations*, 2018.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*, 2020.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

Andrew M Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *International Conference on Machine Learning*, 2011.

Tim Seyde, Wilko Schwarting, Sertac Karaman, and Daniela Rus. Learning to plan optimistically: Uncertainty-guided deep exploration via latent model ensembles, 2021. URL https://openreview.net/forum?id=vT0NSQlTA.

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.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *Internatial Conference on Machine Learning*, 2020.

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

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

Adrien Ali Taïga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. Benchmarking bonus-based exploration methods on the arcade learning environment. In *International Conference on Learning Representations*, 2020.

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. In *Advances in Neural Information Processing Systems*, 2017.

Ruo Yu Tao, Vincent François-Lavet, and Joelle Pineau. Novelty search in representational space for sample efficient exploration. In *Advances in Neural Information Processing Systems*, 2020.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control. *arXiv preprint arXiv:2006.12983*, 2020.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Santosh S Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005.

John Wieting and Douwe Kiela. No training required: Exploring random encoders for sentence classification. In *International Conference on Learning Representations*, 2019.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy, 2010.

Appendix

A Source Code and Videos

We provide source code that reproduces our experimental results and videos in Figure 7a in https://anonymous.4open.science/r/3f955657-271b-4e5d-9a66-f77ecc3708de/.

B Details on DeepMind Control Suite Experiments

B.1 Environments

We evaluate the performance of RE3 on various tasks from DeepMind Control Suite (Tassa et al., 2020). For Hopper Hop, Quadruped Run, Cartpole Swingup Sparse, Pendulum Swingup, we use the publicly available environments without any modification. For environments which are not from the publicly available released implementation repository (https://github.com/deepmind/dm_control), we designed the tasks following Seyde et al. (2021) as below:

- **Walker/Cheetah Run Sparse**: The goal of Walker/Cheetah Run Sparse task is same as in Walker/Cheetah Run, moving forward as fast as possible, but reward is given sparsely until it reaches a certain threshold: $r = r_{\texttt{original}} \cdot \mathbb{1}_{r_{\texttt{original}} > 0.25}$, where $r_{\texttt{original}}$ is the reward in Walker/Cheetah Run from DeepMind Control Suite.

B.2 Implementation Details for Model-free RL

For all experimental results in this work, we report the results obtained by running experiments using the publicly available released implementations from the authors[2], i.e., RAD (https://github.com/MishaLaskin/rad) and DrQ (https://github.com/denisyarats/drq). We use random crop augmentation for RAD and random shift augmentation for DrQ. We provide a full list of hyperparameters in Table 1.

Implementation details for RE3. We highlight key implementation details for RE3:

- **Intrinsic reward.** We use $r^{\texttt{i}}(s_i) := ||y_i - y_i^{k\text{-NN}}||_2$ for the intrinsic reward. We got rid of $\log$ from intrinsic reward in (3) for simplicity in DeepMind Control Suite experiments, but results using $\log$ are also similar. To make the scale of intrinsic reward $r^{\texttt{i}}$ consistent across tasks, following Liu & Abbeel (2021), we normalize the intrinsic reward by dividing it by a running estimate of the standard deviation. As for newly introduced hyperparameters, we use $k = 3$ and $\beta_0 = 0.05$, and perform hyperparameter search over $\rho \in \{0.00001, 0.000025\}$.
- **Architecture.** For all model-free RL methods, we use the same encoder architecture as in Yarats et al. (2019). Specifically, this encoder consists of 4 convolutional layers followed by ReLU activations. We employ kernels of size 3×3 with 32 channels for all layers, and 1 stride except of the first layer which has stride 2. The output of convolutional layers is fed into a single fully-connected layer normalized by LayerNorm (Ba et al., 2016). Finally, tanh nonlinearity is added to the 50-dimensional output of the fully-connected layer.
- **Unsupervised pre-training.** For unsupervised pre-training, we first train a policy to maximize intrinsic rewards in (3) without extrinsic rewards for 500K environment steps. For fine-tuning a policy in downstream tasks, we initialize the parameters using the pre-trained parameters and then learn a policy to maximize RE3 objective in (4) for 500K environment steps. To stabilize the initial fine-tuning phase by making the scale of intrinsic reward consistent across pre-training and fine-tuning, we load the running estimate of the standard deviation from pre-training phase.

Implementation details for representation learning baselines . We highlight key implementation details for state entropy maximization schemes that involve representation learning, i.e., contrastive learning and inverse dynamics prediction, and that employs a pre-trained ImageNet Encoder:

[2]We found that there are some differences from the reported results in the original papers which are due to different random seeds. We provide full source code and scripts for reproducing the main results to foster reproducibility.

- **Contrastive learning.** For state entropy maximization with contrastive learning, we introduce a separate convolutional encoder with randomly initialized parameters learned to minimize contrastive loss (Srinivas et al., 2020). Note that we use the separate encoder to investigate the independent effect of representation learning. Then we compute intrinsic reward $r^{\mathtt{i}}$ using representations obtained by processing observations through this separate encoder.
- **Inverse dynamics prediction.** For state entropy maximization with inverse dynamics prediction, we introduce a separate convolutional encoder with randomly initialized parameters learned to predict actions from two consecutive observations. Specifically, we introduce 2 fully-connected layers with ReLU activations to predict actions on top of encoder representations. We remark that this separate encoder is separate from RL. Then we compute intrinsic reward $r^{\mathtt{i}}$ using representations obtained by processing observations through this separate encoder.
- **Pre-trained ImageNet encoder.** For state entropy maximization with pre-trained ImageNet encoder, we utilize a ResNet-50 (He et al., 2016) encoder from publicly available torchvision models (`https://pytorch.org/vision/0.8/models.html`). To compute intrinsic reward $r^{\mathtt{i}}$, we utilize representations obtained by processing observations through this pre-trained encoder.
- **Pre-trained ATC encoder.** For state entropy maximization with pre-trained ATC (Stooke et al., 2020) encoder, we utilize pre-trained encoders from the authors. Specifically, these encoders are pre-trained by contrastive learning on pre-training datasets that contain samples encountered while training a RAD agent on DeepMind Control Suite environments (see Stooke et al. (2020) for more details). We use pre-trained parameters from Walker Run, Hopper Stand, and Cheetah Run for RAD + SE w/ ATC on Walker Run Sparse, Hopper Hop, and Cheetah Run Sparse, respectively. To compute intrinsic reward $r^{\mathtt{i}}$, we utilize representations obtained by processing observations through this pre-trained encoder.

Implementation details for exploration baselines. We highlight key implementation details for exploration baselines, i.e., RND (Burda et al., 2019) and ICM (Pathak et al., 2017):

- **RND.** For RND, we introduce a random encoder f_θ whose architecture is same as in Yarats et al. (2019), and introduce a predictor network g_ϕ consisting of a convolutional encoder with the same architecture and 2-layer fully connected network with 1024 units each. Then, parameters ϕ of the predictor network are trained to predict representations from a random encoder given the same observations, i.e., minimize $\epsilon = ||f_\theta(s_i) - g_\phi(s_i)||_2$. We use prediction error ϵ as an intrinsic reward and learn a policy that maximizes $r^{\mathtt{total}} = r^{\mathtt{e}} + \beta \cdot r^{\mathtt{i}}$. We perform hyperparameter search over the weight $\beta \in \{0.05, 0.1, 1.0, 10.0\}$ and report the best result on each environment.
- **ICM.** For ICM, we introduce a convolutional encoder g_ϕ whose architecture is same as in Yarats et al. (2019), and introduce a inverse dynamics predictor h_ψ with 2 fully-connected layers with 1024 units. These networks are learned to predict actions between two consecutive observations, i.e., minimize $\mathcal{L}_{\mathtt{inv}} = ||a_t - h_\psi(g_\phi(s_t), g_\phi(s_{t+1}))||_2$. We also introduce a forward dynamics predictor f_Ψ that is learned to predict the representation of next time step, i.e., minimize $\mathcal{L}_{\mathtt{forward}} = \frac{1}{2}||g_\phi(s_{t+1}) - f_\Psi(g_\phi(s_t), a_t)||_2^2$. Then, we use prediction error as an intrinsic reward and learn a policy that maximizes $r^{\mathtt{total}} = r^{\mathtt{e}} + \beta \cdot r^{\mathtt{i}}$. For joint training of the forward and inverse dynamics, following Pathak et al. (2017), we minimize $0.2 \cdot \mathcal{L}_{\mathtt{forward}} + 0.8 \cdot \mathcal{L}_{\mathtt{inv}}$. We perform hyperparameter search over the weight $\beta \in \{0.05, 0.1, 1.0\}$ and report the best result on each environment.

B.3 Implementation Details for Model-based RL.

For Dreamer, we use the publicly available released implementation repository (`https://github.com/danijar/dreamer`) from the authors[3]. We highlight key implementation details for the combination of Dreamer with RE3:

- **Intrinsic reward.** We use $r^{\mathtt{i}}(s_i) := ||y_i - y_i^{k\text{-NN}}||_2$ for the intrinsic reward. We got rid of $\log$ from intrinsic reward in (3) for simplicity in DeepMind Control Suite experiments, but

[3]We use the newer implementation of Dreamer following the suggestion of the authors, but we found that there are some difference from the reported results in the original paper which are might be due to the difference between newer implementation and the original implementation, or different random seeds. We provide full source code and scripts for reproducing the main results to foster reproducibility.

results using log are also similar. To make the scale of intrinsic reward $r^{\mathtt{i}}$ consistent across tasks, following Liu & Abbeel (2021), we normalize the intrinsic reward by dividing it by a running estimate of the standard deviation. Since Dreamer utilizes trajectory segments for training batch, we use large value of $k = 50$ to avoid find k-NN only within a trajectory segment. We also use $\beta_0 = 0.1$, and $\rho = 0.0$, i.e., no decay schedule. We also remark that we find k-NN only within a minibatch instead of the entire buffer as in model-free RL, since the large batch size of Dreamer is enough for stable entropy estimation.

- **Architecture.** We use the same convolutional architecture as in Dreamer. Specifically, this encoder consists of 4 convolutional layers followed by ReLU activations. We employ kernels of size 4×4 with $\{32, 64, 128, 256\}$ channels, and 2 stride for all layers. To obtain low-dimensional representations, we additionally introduce a fully-connected layer normalized by LayerNorm (Ba et al., 2016). Finally, tanh nonlinearity is added to the 50-dimensional output of the fully-connected layer.

Table 1: Hyperparameters of RAD + RE3 used for DeepMind Control Suite experiments.

Hyperparameter	**Value**
Augmentation	Crop
Observation rendering	$(100, 100)$
Observation downsampling	$(84, 84)$
Replay buffer size	100000
Initial steps	10000 quadruped, run; 1000 otherwise
Stacked frames	3
Action repeat	4 quadruped, run; 2 otherwise
Learning rate (actor, critic)	0.0002
Learning rate (α)	0.001
Batch size	512
Q function EMA τ	0.01
Encoder EMA τ	0.05
Critic target update freq	2
Convolutional layers	4
Number of filters	32
Latent dimension	50
Initial temperature	0.1 RAD + RE3; 0.01 RAD + RE3 + PT
k	3
Initial intrinsic reward scale β_0	0.05
Intrinsic reward decay ρ	0.000025 walker, run sparse; 0.00001 otherwise
Intrinsic reward scale (RND)	10.0 cartpole, swingup sparse 0.1 pendulum, swingup; cheetah, run sparse 0.05 otherwise
Intrinsic reward scale (ICM)	1.0 cheetah, run sparse 0.1 otherwise

Table 2: Hyperparameters of Dreamer + RE3 used for DeepMind Control Suite experiments. We only specify hyperparameters different from original paper of Hafner et al. (2020).

Hyperparameter	**Value**
Initial episodes	5
Precision[4]	32
Latent dimension of a random encoder	50
k	53
Initial reward scale β_0	0.1
Intrinsic reward decay ρ	0.0

[4]We found that precision of 32 is necessary for avoiding NaN in intrinsic reward normalization, as running estimate of standard deviation is very small.

C ADDITIONAL EXPERIMENTAL RESULTS ON DEEPMIND CONTROL SUITE

Comparative evaluation. We provide additional experimental results on various tasks from DeepMind Control Suite in Figure 9. We observe that RE3 improves sample-efficiency in several tasks, e.g., Reacher Hard and Hopper Stand, while not degrading the performance in dense-reward tasks like Cartpole Balance. This demonstrates the simple and wide applicability of RE3 to various tasks.

Comparison with other exploration methods. Figure 10 shows that RAD + RE3 consistently exhibits superior sample efficiency to other exploration methods (i.e., RAD + RND, RAD + ICM) on additional tasks from DeepMind Control Suite.

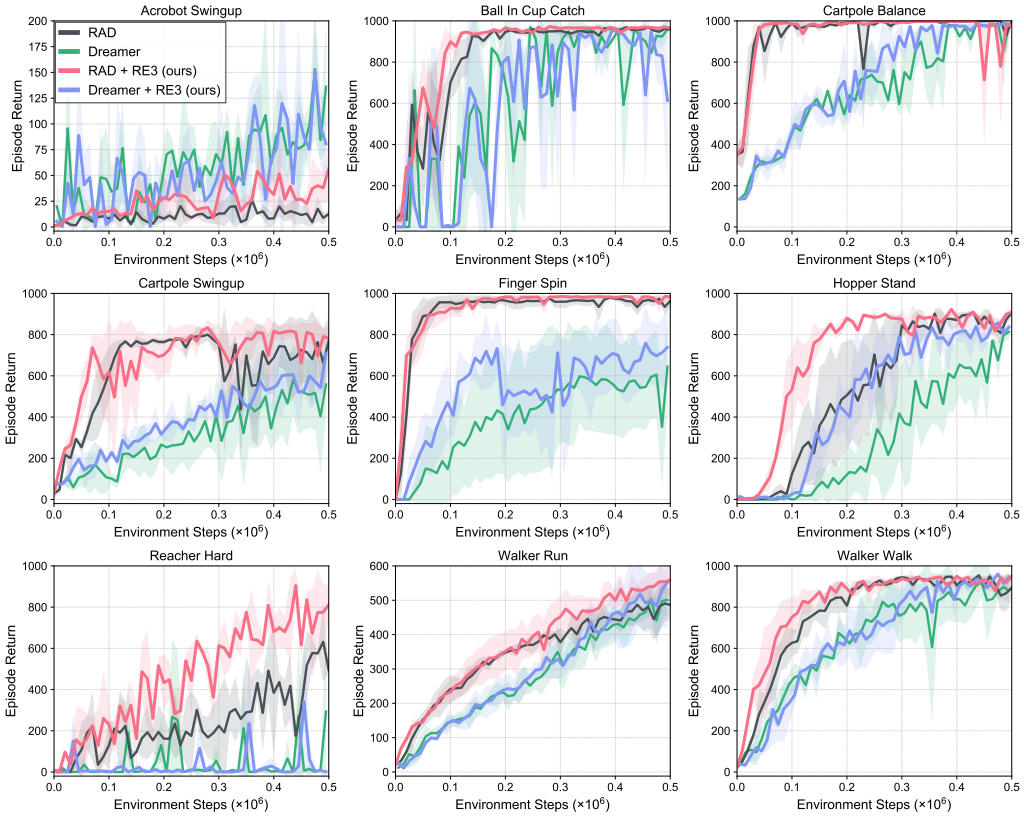


Figure 9: Performance on locomotion tasks from DeepMind Control Suite. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.

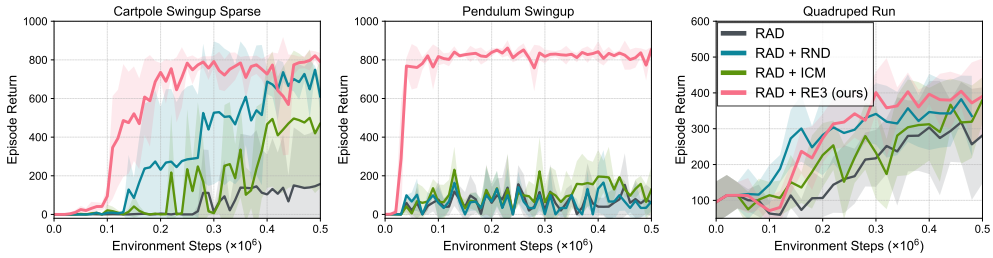


Figure 10: Performance on locomotion tasks from DeepMind Control Suite. RAD + RE3 outperforms other exploration methods in terms of sample-efficiency. The solid line and shaded regions represent the mean and standard deviation, respectively, across five runs.

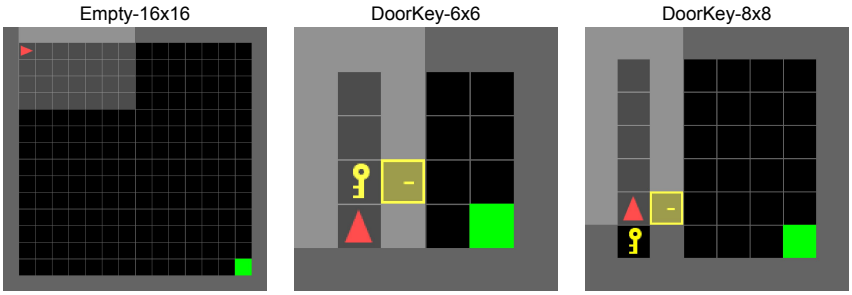


Figure 11: Navigation tasks from MiniGrid (Chevalier-Boisvert et al., 2018) used in our experiments. The agent is represented as a red arrow and the light gray region shows the 7×7 (or smaller, if obstructed by walls) grid which the agent observes. The agent receives a positive reward only for reaching the green square.

D Details on MiniGrid Experiments

For our A2C implementation in MiniGrid, we use the publicly available released implementation repository (https://github.com/lcswillems/rl-starter-files) and use their default hyperparameters. We provide a full list of hyperparameters that are introduced by our method or emphasized for clarity in Table 3.

We highlight some key implementation details:

- We use $r^{\mathtt{i}}(s_i) := \log(||y_i - y_i^{k\text{-NN}}||_2 + 1)$ for the intrinsic reward. The additional 1 is for numerical stability.
- We use the average distance between y_i and its k nearest neighbors (i.e., $y_i^{2\text{-NN}}, \cdots, y_i^{k\text{-NN}}$) for the intrinsic reward, instead of the single k nearest neighbor. This provides a less noisy state entropy estimate and empirically improves performance in MiniGrid environments.
- We perform hyperparameter search over $\beta \in \{0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$ for RE3, ICM, and RND and report the best result.
- We do not change the network architecture from the above publicly available implementation. We use the same encoder architecture as the RL encoder for state entropy maximization. This encoder architecture consists of 3 convolutional layers with kernel 2, stride 1, and padding 0, each followed by a ReLU layer. The convolutional layers have 16, 32, and 64 filters respectively. The first ReLU is followed by a two-dimensional max pooling layer with kernel 2. The actor and critic MLPs both contain two fully-connected layers with hidden dimension 64, with a tanh operation in between.

Table 3: Hyperparameters used for MiniGrid experiments. Most hyperparameter values are unchanged across environments with the exception of evaluation frequency and intrinsic reward weight β.

Hyperparameter	**Value**
Input Size	$(7, 7, 3)$
Replay buffer size (for RE3 intrinsic reward)	10000
Stacked frames	1
Action repeat	1
Evaluation episodes	100
Optimizer	RMSprop
k	3
Evaluation frequency	6400 Empty-16x16; 12800 DoorKey-6x6 64000 DoorKey-8x8
Intrinsic reward weight β in Empty-16x16 experiments	0.1 A2C + RE3 + PT 0.1 A2C + RE3 0.00001 A2C + ICM 0.00005 A2C + RND
Intrinsic reward weight β in DoorKey-6x6 experiments	0.05 A2C + RE3 + PT 0.005 A2C + RE3 0.0001 A2C + ICM 0.0001 A2C + RND
Intrinsic reward weight β in DoorKey-8x8 experiments	0.05 A2C + RE3 + PT 0.01 A2C + RE3 0.001 A2C + ICM 0.00005 A2C + RND
Intrinsic reward decay ρ	0
Number of processes	16
Frames per process	5
Discount γ	0.99
GAE λ	0.95
Entropy coefficient	0.01
Value loss term coefficient	0.5
Maximum norm of gradient	0.5
RMSprop ϵ	0.01
Clipping ϵ	0.2
Recurrence	None

E CALCULATION OF FLOATING POINT OPERATIONS

We explain our FLOP counting procedure for comparing the compute-efficiency of RAD, RAD + SE w/ Random (ours), RAD + SE w/ Contrastive, and RAD + SE w/ Inverse dynamics in Figure 7. We consider each backward pass to require twice as many FLOPs as a forward pass, as done in https://openai.com/blog/ai-and-compute/. Each weight requires one multiply-add operation in the forward pass. In the backward pass, it requires two multiply-add operations: at layer i, the gradient of the loss with respect to the weight at layer i and with respect to the output of layer $(i-1)$ need to be computed. The latter computation is necessary for subsequent gradient calculations for weights at layer $(i-1)$.

We use functions from Huang et al. (2018) and Jeong & Shin (2019) to obtain the number of operations per forward pass for all layers in the encoder (denoted E) and number of operations per forward pass for all MLP layers (denoted M).

We assume that (1) the number of updates per iteration is 1, (2) the architecture of the encoder used for state entropy estimation is the same as the RL encoder used in RAD, and (3) the FLOPs required for computations (e.g., finding the k-NNs in representation space) that are not forward and backward passes through neural network layers is negligible.[5]

We denote the number of forward passes per training update F, the number of backward passes per training update B, and the batch size b (in our experiments $b = 512$). Then, the number of FLOPs per iteration of RAD is:

$$bF(E+M) + 2bB(E+M) + (E+M),$$

where the last term is for the single forward pass required to compute the policy action.

Specifically, RAD + SE w/ Random only requires E extra FLOPs per iteration to store the fixed representation from the random encoder in the replay buffer for future k-NN calculations. In comparison, RAD + SE w/ Contrastive requires $4bE$ extra FLOPs per iteration, and RAD + SE w/ Inverse dynamics requires more than $6bE$ extra FLOPs.

[5]Letting the dimension of y be d, batch size m, computing distances between y_i and all entries $y \in \mathcal{B}$ over 250000 training steps requires $\sum_{n=1000}^{250000}(d(2m + 2 \cdot \min(n, |\mathcal{B}|) + 3m \cdot \min(n, |\mathcal{B}|)) + 2m \cdot \min(n, |\mathcal{B}|))$, which is 1.569e+15 FLOPs in our case of $m = 512$, $|\mathcal{B}| = 100000$, and $d = 50$.

F COMPARISON TO STATE ENTROPY MAXIMIZATION WITH CONTRASTIVE ENCODER FOR MINIGRID PRE-TRAINING

We show a comparison to state entropy maximization with contrastive learning (i.e., A2C + SE w/ Contrastive + PT) in Figure 12. In the results shown, we do not use state entropy intrinsic reward during the fine-tuning phase for A2C + SE w/ Contrastive + PT, following the setup of Liu & Abbeel (2021), but found that using intrinsic reward during fine-tuning results in very similar performance. As discussed in Section 4.2, we observe that the contrastive encoder does not work well for state entropy estimation, as contrastive learning depends on data augmentation specific to images (e.g., random shift, random crop, and color jitter), which are not compatible with the compact embeddings used as inputs for MiniGrid. We re-mark that RE3 eliminates the need for carefully chosen data augmentations by employing a random encoder.

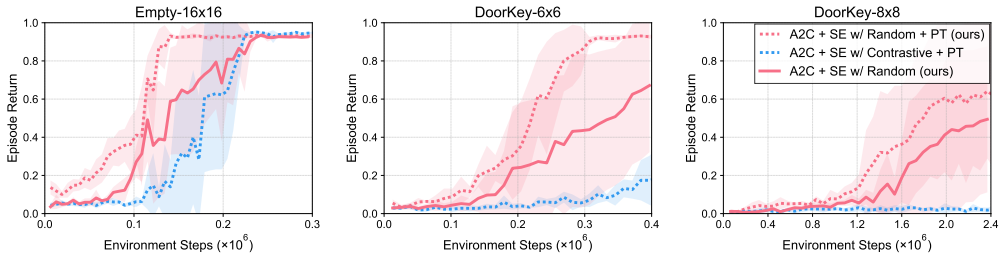


Figure 12: Performance on navigation tasks from MiniGrid. We find that pre-training using state entropy (SE) with a random encoder (ours) outperforms pre-training using state entropy with a contrastive encoder (Liu & Abbeel, 2021), using random shift as the data augmentation. The solid (or dotted) line and shaded regions represent the mean and standard deviation, respectively, across five runs.

G RE3 WITH ON-POLICY REINFORCEMENT LEARNING

We provide the full procedure for RE3 with on-policy RL in Algorithm 2.

Algorithm 2 RE3: On-policy RL version

Initialize parameters of random encoder θ, policy ϕ
Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$, step counter $t \leftarrow 0$
repeat
  $t_{\texttt{start}} \leftarrow t$ // COLLECT TRANSITIONS
  repeat
    Collect a transition $\tau_t = (s_t, a_t, s_{t+1}, r_t^{\texttt{e}})$ using policy π_ϕ
    Get a fixed representation $y_t = f_\theta(s_t)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\tau_t, y_t)\}$
    $t \leftarrow t + 1$
  until terminal s_t or $t - t_{\texttt{start}} = t_{\texttt{max}}$
  for $j = t - 1$ **to** $t_{\texttt{start}}$ **do** // COMPUTE INTRINSIC REWARD
    Compute the distance $||y_j - y||_2$ for all $y \in \mathcal{B}$ and find the k-nearest neighbor $y_j^{k\text{-NN}}$
    Compute $r_j^{\texttt{i}} \leftarrow \log(||y_j - y_j^{k\text{-NN}}||_2 + 1)$
    Let $r_j^{\texttt{total}} \leftarrow r_j^{\texttt{e}} + \beta \cdot r_j^{\texttt{i}}$
  end for
  Update ϕ with transitions $\{(s_j, a_j, s_{j+1}, r_j^{\texttt{total}})\}_{j=t_{\texttt{start}}}^{t-1}$
until $t > t_{\texttt{max}}$ // UPDATE POLICY