Novelty-Guided Reinforcement Learning via Encoded Behaviors

Rajkumar Ramamurthy
Fraunhofer IAIS
Sankt Augustin, Germany

Rafet Sifa
Fraunhofer IAIS
Sankt Augustin, Germany

Max Lübbering
Fraunhofer IAIS
Sankt Augustin, Germany

Christian Bauckhage
Fraunhofer Center for ML
Sankt Augustin, Germany

Abstract—**Despite the successful application of Deep Reinforcement Learning (DRL) in a wide range of complex tasks, agents either often learn sub-optimal behavior due to the sparse/deceptive nature of rewards or require a lot of interactions with the environment. Recent methods combine a class of algorithms known as Novelty Search (NS), which circumvents this problem by encouraging exploration towards novel behaviors. Even without exploiting any environment rewards, they are capable of learning skills that yield competitive results in several tasks. However, to assign novelty scores to policies, these methods rely on neighborhood models that store behaviors in an archive set. Hence they do not scale and generalize to complex tasks requiring too many policy evaluations. Addressing these challenges, we propose a function approximation paradigm to instead learn sparse representations of agent behaviors using auto-encoders, which are later used to assign novelty scores to policies. Experimental results on benchmark tasks suggest that this way of novelty-guided exploration is a viable alternative to classic novelty search methods.**

Index Terms—**Reinforcement Learning, Exploration, Novelty Search**

I. Introduction

One of the goals of artificial intelligence is to develop agents that learn to perform any given task by interacting with an unknown environment. Recent progress in this regard has been achieved through reinforcement learning techniques in combination with deep learning (DRL). These were applied successfully to a wide range of complex tasks including playing games [14], [31], navigating complex environments [13], [35], and controlling robots [6], [28]. Nevertheless, RL faces several challenges that limit its application in real-world systems. First of all, the learning is purely based on sparse reward signals which are available only when agents reach the goal state. Especially in tasks dealing with long horizons, propagating reward signals to past actions poses a significant challenge of temporal credit assignment and requires a large number of roll-outs to learn a suitable policy. Addressing this challenge, a multitude of methods have been proposed including shaping rewards [15], curriculum learning [7] and temporal difference methods that extend bootstrapping using multi-step returns [2], [29].

Second of all, for many real-world problems, the design of reward functions can be challenging and susceptible to several concerns [1]; (i) *reward hacking*: the agent may maximize the reward without performing the intended task. For example, in a system that learns to ride bicycle [23], the agent had learned to ride in circles around its starting position because the reward function was chosen such that positive reward is given every time progress is made towards the goal but, on the other hand, no penalty was given for not reaching the goal position (ii) *deceptiveness*: the reward functions can be deceptive. Consider, for example, an agent learning to navigate in a grid environment with a reward function that rewards positions close to the goal position. Simple policy learning without adequate exploration may get stuck in deceptive walls and fail to reach the target. Inverse reinforcement learning methods [16], [33] tackle this by inferring reward functions directly from expert demonstrations while simultaneously learning a policy for solving the given task. Alternatively, approaches that involve human interaction [11], [20] rely on obtaining preferences about policies from human trainers to learn desired optimal behavior.

Third of all, agents interacting in RL environments encounter the *exploration-exploitation* dilemma. If the agent greedily takes actions with high action-value, it may fail to uncover alternate actions that may yield better payoffs later on. In several RL solutions, exploration is still mostly random and performed using simple methods such as epsilon-greedy strategies, or Gaussian or Boltzmannian policies. While approaches such as count-based exploration [17], intrinsic or curiosity [18] have been studied recently, they either require enumerating over state-action spaces or require learning state-transition models rendering them impractical for high-dimensional tasks. Alternatively, exploration is also achieved in parameter space of deep neural networks using black-box evolutionary algorithms such as Evolution Strategies (ES) [25], [26] which have been found to have faster training wall-clock times and scale better than traditional RL algorithms. Yet, deceptiveness and sparseness in reward signals still cause longer training times in ES thus demanding a more directed exploration instead of random exploration.

This work belongs to the class of methods known as Novelty Search (NS) [9], [10] which are well suited to promote directed exploration in sparse/deceptive problems. The principal idea is to incorporate a domain-specific behavioral characteristic (BC) that captures agent's behavior and then to encourage the agent to perform different (novel) behaviors than exhibited earlier. For instance, [4] considers "final position of agent" as the BC while [9] uses "distance travelled by agent" in a bipedal locomotion task and found that optimizing novelty of this BC is superior to optimizing the standard reward objective. In general, BCs can be defined by domain experts based on the task at hand directly leveraging expert knowledge. Our work is based on an approach [4] that integrates a novelty objective into the standard reward objective which has shown to improve exploration in several benchmark tasks. However, such an approach has several drawbacks. Typically, an archive set of fixed size is used to store observed policies and their behaviors. To compute novelty of a given policy, its nearest neighbors are first retrieved and a simple metric then assigns scores to the given policy based on distance to its neighbors. As complex tasks require a large dimensional behavior characteristic and evaluate a large number of policies, finding nearest neighbors on such a large archive set becomes a bottleneck computationally. Also, it may not generalize well as only a limited set of policies can be stored in the archive set.

In this work, we alleviate these concern of scalability and generalizability pertaining to novelty search methods. In particular, we propose to take a function approximation perspective and employ an auto-encoder to learn representation of agent behaviors. Instead of using distance based methods for computing novelty, they can be encoded using auto-encoders. Then, reconstruction errors can be used as novelty bonuses to policies. The main idea is that novel behaviors tend to produce relatively larger errors than behaviors observed in the past. This way of assigning novelty scores to policies based on their reconstruction error can encourage robust exploration towards less visited areas of behavior space. Further, in order to encode diverse set of behaviors and to encourage generalization, we propose to use a sparse variant of auto-encoder namely k-sparse auto-encoders. More importantly, we learn this model concurrently along with the learning policy. Therefore, our main contribution is a simple, scalable and efficient exploration approach using novelty scores for policies along with a detailed experimental evaluation against classic novelty search and policy gradient methods.

II. Related Work

Exploration is an actively studied area in reinforcement learning. Several methods have been proposed to promote directed exploration in RL; The general theme of most approaches is to encourage agent to visit states that are seldom visited. Early work [27] in this regard proposed to learn a curiosity model which predicts future events using history of interactions with environment. The arising prediction error is then seen as intrinsic reward that drives towards creative solutions. Methods [3], [17] that assign novelty scores based on visitation counts either on raw states or encoded states were also proposed. On the other hand, approaches that maximize the information theoretic objectives [5], [8] to learn exploration strategy by encouraging diversity. Methods that are most related to our approach are convolutional auto-encoders [34] to encode the given observation to hash codes upon which count-based exploration rewards are generated, forward dynamics model [18], [32] that learn one-step state dynamics from previous states and actions; and use prediction error as exploration rewards. Extending on previous methods, an interesting work [19] proposed to use disagreement arising from an ensemble of forward models as intrinsic rewards to policy learning. While most previous work considered generating exploration bonuses at each step using input observation space, in contrast, our work focuses on encoding episodic agent behaviors in novelty search methods.

III. Preliminaries

A. Markov Decision Process

Typically, a standard reinforcement learning setting of an agent interacting with an environment; The environment is formulated as an infinite-horizon discounted Markov Decision Process (MDP) defined by a tuple $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{R}, \lambda \rangle$ where $\boldsymbol{S}$ is a set of states, $\boldsymbol{A}$ is a set of actions available to the agent, $\boldsymbol{R}$ is the reward function, $\boldsymbol{T}$ is the transition probability. and $\lambda \in (0, 1)$ is a discount factor. At each time step t, the agent receives an observation $\boldsymbol{o_t}$ about the state $\boldsymbol{s_t}$ of the environment, performs an action $\boldsymbol{a_t}$, and receives a scalar reward r_t. Upon an action $\boldsymbol{a_t}$, the environment transitions to a new state $\boldsymbol{s_{t+1}}$ according to the function $\boldsymbol{T}(\boldsymbol{s_{t+1}}, \boldsymbol{s_t}, \boldsymbol{a_t})$ and returns with a scalar reward $r_t = \boldsymbol{R}(\boldsymbol{s_{t+1}}, \boldsymbol{s_t}, \boldsymbol{a_t})$.

Planning in MDP is achieved by following a policy function $\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ which maps each state-action pair $(\boldsymbol{s_t}, \boldsymbol{a_t})$ to the probability of selecting the action in the particular state; Therefore, our goal is to find an optimal policy that maximizes the return discounted by $\lambda \in (0, 1)$ over a period of time T given as $G_T = \sum_{t=1}^{T} \lambda^{t-1} r_t$. For high-dimensional state and action spaces, the policy π is often represented as a deep neural network $\pi_{\boldsymbol{\theta}}$ with weights $\boldsymbol{\theta}$. Then, the goal is to determine optimal weights $\boldsymbol{\theta}^*$ that maximize the expected cumulative reward $\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\pi_\theta} \big[G_T \big]$

The policy learning is typically achieved via stochastic gradient ascent where the gradient $\nabla_\theta \mathbb{E}_{\pi_\theta} \big[G_T \big]$ is obtained using sampled sequences $(\boldsymbol{s_t}, \boldsymbol{a_t}, r_t, \boldsymbol{s_{t+1}} \dots)$ of interactions with the environment and can be computed via policy gradient methods [21], [22], [26], [28], [30]. In our work, we consider to use a gradient approximation technique described in the next section to estimate this gradient rather than using online policy gradient algorithms. This choice is primarily due to its benefit of easy integration of novelty scores into the objective function as well as its parallelizable capabilities which is well studied in [4], [26].

B. Evolution Strategies (ES)

Evolution Strategies (ES) [24], [25] are heuristic search procedures inspired by evolution. In each iteration, a population of perturbed parameters is generated and an objective function is evaluated. Using a process akin to natural selection, parameter vectors are then combined to create the next population and this process continues until a a population reaches a satisfactory performance. There exist several flavors of ES w.r.t. the representation of parameters or the selection process. The version we use here belong to the class of Natural Evolution Strategies (NES). Let f be the objective function acting upon parameters $\boldsymbol{\theta}$. In a reinforcement learning setting, it is the stochastic return obtained from the environment. NES algorithms maintain the population as a distribution over parameters $\boldsymbol{\theta}$. Typically, the distribution corresponds to a multivariate Gaussian centered around the current parameter with co-variance $\sigma^2 I$ (ie.) $\boldsymbol{\theta} \sim \mathcal{N}(0, \sigma^2 I)$. Given this, NES seeks to maximize the average objective of the population $\mathbb{E}_{\boldsymbol{\theta} \sim \mathcal{N}(0,\sigma^2 I)}[F(\boldsymbol{\theta})]$ by optimizing the parameters $\boldsymbol{\theta}$. Generally, NES also updates the co-variance of the population distribution, but as in other RL approaches, we use a static co-variance σ by fixing it throughout the training.

To estimate the gradient of the expected cumulative reward in iteration k, n perturbations are sampled from the distribution by adding Gaussian noise to the current parameter vector $\boldsymbol{\theta}$ (i.e. $\boldsymbol{\theta}_k = \boldsymbol{\theta}_k + \sigma \boldsymbol{\epsilon}_i$ where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, I)$). The gradient is then approximated by a sum of sampled perturbations weighted by their corresponding objective function measurements

$$\nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[F(\boldsymbol{\theta}_k + \sigma \boldsymbol{\epsilon})] \approx \frac{1}{n\sigma} \sum_{i=1}^{n} F(\boldsymbol{\theta}_k^i) \boldsymbol{\epsilon}_i \tag{1}$$

Typically, the objective function evaluations $F(\boldsymbol{\theta}_k^i)$ are subject to rank-normalization before computing the gradients to ensure that the reward scales between different tasks does not affect the optimization process.

C. Novelty Search with Nearest Neighbors

One of the reasons why RL can not cope well with sparse/deceptive problems is that the reward function usually does not take into account intermediate stepping stones that would allow for learning target skills. Hence, solving such tasks purely with goal-only rewards raises considerable challenges and demands numerous interactions with environment. Novelty Search (NS), inspired by nature's tendency towards evolving increasingly complex behaviors, tackles this by using novelty as a proxy for stepping stones. In other words, NS aims to drive the search process towards policies with higher novelty, rather than to ones with higher cumulative reward.

In order to differentiate behaviors, each policy $\pi_{\boldsymbol{\theta}}$ (parameterized by a DNN with weights $\boldsymbol{\theta}$) is assigned a domain-dependent Behavior Characteristic (BC) denoted as $\boldsymbol{b}(\pi_{\boldsymbol{\theta}})$. For instance, in case of grid navigation or bi-pedal walking domains, it simply could be the final two-dimensional position of the agent at the end of an episode. However, considering only the final position may not be sufficient to distinguish different behaviors terminating at the same position, therefore it is necessary capture the trajectory of agent positions concatenated as a sequence to form BC [4].

In the classical sense, a set of fixed size typically known as archive set $\boldsymbol{A}$ is maintained to store observed policies and their behaviors. Given this set, we desire a metric $N(\boldsymbol{\theta}, \boldsymbol{A})$ that allows us to measure novelty of a given parameterized policy $\pi_{\boldsymbol{\theta}}$. Typically, it is defined as the average distance to its nearest neighbors $\boldsymbol{K}$. Higher the distance to its neighbors, higher is the novelty and vice versa.

$$N(\boldsymbol{\theta}, A) = \frac{1}{|\boldsymbol{K}|} \sum_{i \in \boldsymbol{K}} \|\boldsymbol{b}(\pi_{\boldsymbol{\theta}}) - \boldsymbol{b}(i)\|^2 \tag{2}$$

$$\nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[N(\boldsymbol{\theta}_k + \sigma \boldsymbol{\epsilon})] \approx \frac{1}{n\sigma} \sum_{i=1}^{n} N(\boldsymbol{\theta}_k^i, \boldsymbol{A}) \epsilon_i \tag{3}$$

We can then resort to the ES framework from Sec III-B to estimate the novelty gradient with respect to current policy parameters $\boldsymbol{\theta}_k$ and taking a step towards parameters that produce novel behaviors. Initially, this scheme of search may start with idle behaviors or ones that fail immediately. As the optimization progresses, these behaviors will become less novel paving way for "stepping stones" to be discovered such as behaviors that walk for few time steps. Further along the optimization, behaviors of increasing complexity are identified, and, eventually, agents learning to walk far or to navigate the grid will be discovered.

D. Combining with RL

Novelty Search can be incorporated in both settings; in sparse/deceptive settings where RL is not just enough and in general to speed up RL by promoting exploration even with dense rewards. In either case, NS and RL objectives can be combined [4] as their weighted combination and the update rule is given as follows:

$$\boldsymbol{\theta_{k+1}} = \boldsymbol{\theta_k} + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} w F(\boldsymbol{\theta}_k^i) \epsilon_i + (1 - w) N(\boldsymbol{\theta}_k^i, \boldsymbol{A}) \epsilon_i \tag{4}$$

where w is the importance given to reward objective, referred as "reward pressure". By following gradients to this objective function, policies that are both novel and achieve higher rewards can be searched. Further, w can be adapted based on learning progress. Initially, w is set to 1.0 to purely pursue environment reward signals. However, if the performance is not improved in a few iterations k_{max}, then w is decreased by δ_w. At this point, gradients slightly start to follow novelty until the performance has improved, then w is incremented by δ_w.

IV. Novelty Bonuses via Encoded Behaviors

Our aim is to come up with a general mechanism for computing novelty of a given policy rather than relying on using neighborhood methods to store and retrieve closest agent behaviors. To that end, we propose to learn representations of agent behaviors such that the prediction error in behavior space provides a good novelty bonus. This representation can be learned using a deep neural network namely an auto-encoder consisting of two components: the first component maps the behavior into an encoded vector and the second component takes the encoded vector as inputs and reconstructs the behavior back. As novel behaviors are unseen inputs to this model, the arising prediction error can be seen as a measure of novelty. In the same way, frequently occurring behaviors tend to have lesser prediction error and will be assigned a lower novelty score.

Aligning with other novelty-search works, we also consider an episodic setting of reinforcement learning (i.e.) novelty bonuses are assigned to a policy at the end of an episode. Therefore, the behavior characteristic is a fixed-length sequence of agent positions sampled at specific intervals as discussed earlier. In addition to novelty rewards, agents also receive the cumulative reward from the environment.

a) ***Training****:* Next, we describe the model of the auto-encoder to learn behavior representations. The model consists of an encoder network with parameters $\boldsymbol{\phi_e}$ and the decoder network with parameters $\boldsymbol{\phi_d}$. The input to the model is a behavior characterization vector $\boldsymbol{b} \in \mathbb{R}^D$. The encoder network projects this vector into an encoded vector $\boldsymbol{h} \in \mathbb{R}^Z$. Given this representation as input, the decoder network re-constructs the input behavior as $\boldsymbol{b}'$ as follows:

$$\boldsymbol{h} = f^e(\boldsymbol{W_e} \cdot \boldsymbol{b} + \boldsymbol{b_e}) \tag{5}$$

$$\boldsymbol{b'} = f^d(\boldsymbol{W_d} \cdot \boldsymbol{h} + \boldsymbol{b_d}) \tag{6}$$

where f^e is the activation function of the encoder which is typically a sigmoid function and f^d is the decoder's activation function which is an identity function. The parameters of the encoder and decoder such as $\boldsymbol{\phi_e} = \{\boldsymbol{W_e}, \boldsymbol{b_e}\}$ and $\boldsymbol{\phi_d} = \{\boldsymbol{W_d}, \boldsymbol{b_d}\}$ respectively are jointly optimized to minimize the reconstruction loss L between the actual and the predicted behaviors as in Eq: (7). In order to perform stable gradient updates, the observed behaviors are stored in a fixed sized behavior buffer (similar to archive set in novelty-search methods). Then, at each learning step, a mini-batch is sampled to perform the gradient descent.

$$L(\boldsymbol{b}, \boldsymbol{b'}) = \sum_{i=1}^{D} (\boldsymbol{b_i} - \boldsymbol{b}'_{\boldsymbol{i}})^2 \tag{7}$$

b) ***Sparse Encoding****:* However, in-order to encode diverse behaviors, we consider k-sparse auto-encoders [12], a simple variant of auto-encoders to enforce sparse representations. During the feed forward phase, the hidden activations of the encoder are sorted and only the top k hidden units are retained while rest of the units are set to zero. By back-propagating only through these active hidden units, the decoder learns to reconstruct the given input by using very few units, thereby also acting as a regularizer. To compare sparsity in networks with different number of hidden units, we define the sparsity level $sparse$ as the ratio of hidden units which are retained for each sample of input. We summarize the training of sparse auto-encoders in the **Algorithm 1**.

c) ***Novelty Scores****:* Given a policy $\pi_{\boldsymbol{\theta}}$ and its behavior characterized by $\boldsymbol{b}(\pi_{\boldsymbol{\theta}})$ obtained by rolling out an episode with $\pi_{\boldsymbol{\theta}}$. Then the novelty bonus is obtained by passing the behavior sequence to encoder-decoder networks to obtain the reconstruction error $N(\boldsymbol{\theta}) = L(\boldsymbol{b}(\pi_{\boldsymbol{\theta}}), \boldsymbol{b}'(\pi_{\boldsymbol{\theta}}))$.

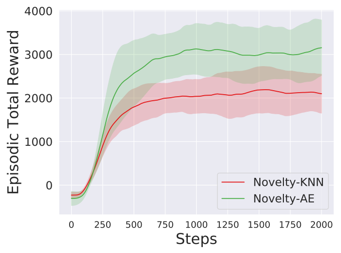
In summary, our agent is composed of two sub-systems: a behavior auto-encoder model that outputs a novelty bonus for the given policy and a policy network that outputs a sequence of actions to maximize the joint objective function of cumulative reward and novelty. We summarize the entire training approach in the **Algorithm 2**.

Algorithm 1 Fit Sparse Behavior Auto-encoders

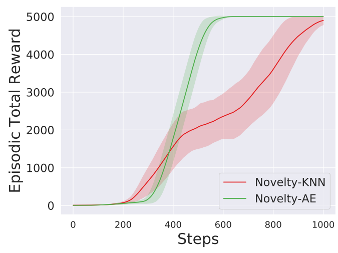
```
Input: Learning rate $\beta$, epochs $E$, batch size $m$, behavior
  auto-encoder parameters $\boldsymbol{\phi_e}$, $\boldsymbol{\phi_d}$, behavior buffer $A$, Sparsity level $sparse$
  for $i = 0$ to $E$ do
    Generate $batches$ of size $m$ from the behavior buffer
    for $\boldsymbol{b}$ in $batches$ do
      Compute encoded vectors $\boldsymbol{h}$ by forwarding $\boldsymbol{b}$ to the encoder using Eq. 5
      Find indices of largest activations of $\boldsymbol{h}$ according to specified sparsity level $\boldsymbol{ind} = top_{sparse}(\boldsymbol{h})$
      Set activation of other units to zero $\boldsymbol{h}(\boldsymbol{ind}^c) = 0$
      Compute decoded values $\boldsymbol{b}'$ using Eq. 6
      Compute the reconstruction error $L(\boldsymbol{b}, \boldsymbol{b}')$ using Eq. 7
      Backpropagate the error $L$ through the encoder and decoder
      Update $\boldsymbol{\phi_e}$ and $\boldsymbol{\phi_d}$ by taking a gradient descent at the rate $\beta$
    end for
  end for
```

Algorithm 2 Novelty-Guided RL via Encoded Behaviors

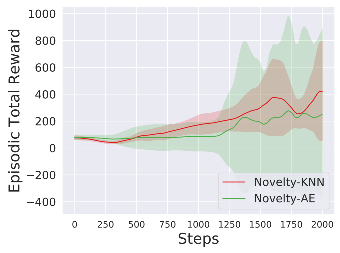
```
Input: Learning rate $\alpha$, initial reward pressure $w$, iterations
  $K$, ES parameters $n$ and $\sigma$
Initialize: Policy parameters $\boldsymbol{\theta}$, behavior auto-encoder, behavior buffer $A = \{\}$
  for $k = 0$ to $K$ do
    for $i = 1$ to $n$ do
      Sample $\epsilon_i \sim \mathcal{N}(0, \sigma^2 I)$
      Compute $\boldsymbol{\theta}_k^i = \boldsymbol{\theta}_k^i + \sigma\boldsymbol{\epsilon}_i$
      Perform a roll-out with policy $\pi_{\boldsymbol{\theta}_k^i}$ and obtain behaviors $b(\boldsymbol{\theta}_k^i)$
      Compute novelty bonus $N(\boldsymbol{\theta}_k^i)$ using behavior auto-encoder's reconstruction error
      Compute cumulative reward $F(\boldsymbol{\theta}_k^i)$
    end for
    Update policy network: $\boldsymbol{\theta_{k+1}} = \boldsymbol{\theta_k} + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} wF(\boldsymbol{\theta}_k^i)\epsilon_i + (1-w)N(\boldsymbol{\theta}_k^i)\epsilon_i$
    Add sampled behaviors $\boldsymbol{b}(\pi_{\boldsymbol{\theta}_k^i})$, $i = 1 \ldots n$ to the behavior buffer
    Update behavior auto-encoder by performing a gradient descent step with behavior buffer
    Adapt the reward pressure $w$ based on learning progress as discussed in Sec III-D
  end for
```



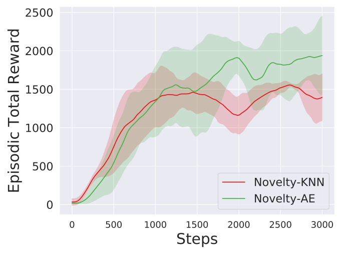
(a) Half Cheetah



(b) Inverted Pendulum

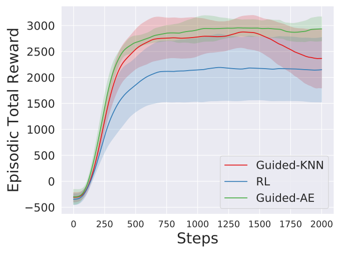


(c) Inverted Double Pendulum

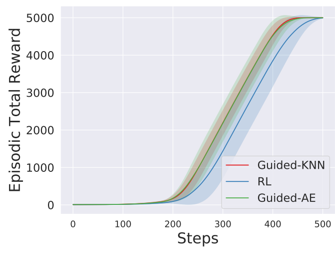


(d) Hopper

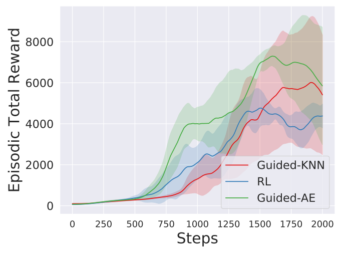
Fig. 1: Average learning curves of agents that are trained using only novelty gradients for the continuous control tasks. We can observe that the pure novelty exploration via encoded behaviors perform better than the classic novelty search methods in most tasks. It is also observed that the agents can still learn to perform the tasks without using the environment rewards



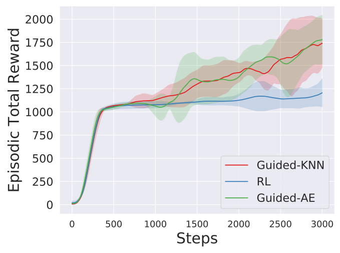
(a) Half Cheetah



(b) Inverted Pendulum



(c) Inverted Double Pendulum



(d) Hopper

Fig. 2: Average learning curves of agents that are trained using both novelty and reward gradients for the continuous control tasks. We can observe that the novelty guided methods perform better than ES using only rewards. In particular, novelty guided methods using autoencoders perform better than classic neighborhood methods.

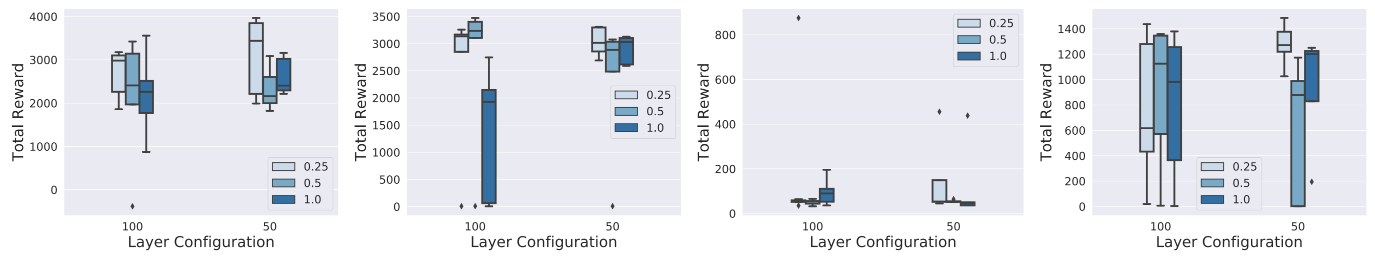
V. Experimental Results

We test our methods on the Mujoco continuous control tasks which provide a standard set of benchmark tasks. In particular, we consider 4 tasks: Inverted Pendulum, Inverted Double Pendulum, Half-Cheetah and Hopper.

For all our experiments, the policy network is a multi-layer perceptron with two hidden layers containing 64 tanh neutrons each. The input to the network is the observation space from the environment and the output is an action vector of motor commands. ES is trained with a learning rate $\alpha = 0.01$ and a noise standard deviation of $\sigma = 0.1$. To keep the experiments tractable, we limit the number of samples drawn from the population distribution in each generation to $n = 50$. For the baseline novelty method using nearest neighbors, we fix $k = 10$ and use an archive set of size 1000 which is implemented as a FIFO so that only the recent behaviors are kept. The behavior characteristic BC is a sequence of agent trajectory sub-sampled at specific intervals to have a fixed length of 50. The behavior auto-encoder is composed of feedforward multi-layer perceptrons whose configurations and their sparsity levels are chosen based on a formal grid search. It is trained with Adam optimizer with learning rate of $\beta = 0.001$ and batch size of $m = 100$. For fair comparison, the size of the behavior buffer is also limited to $|A| = 1000$ containing only the recent behaviors.

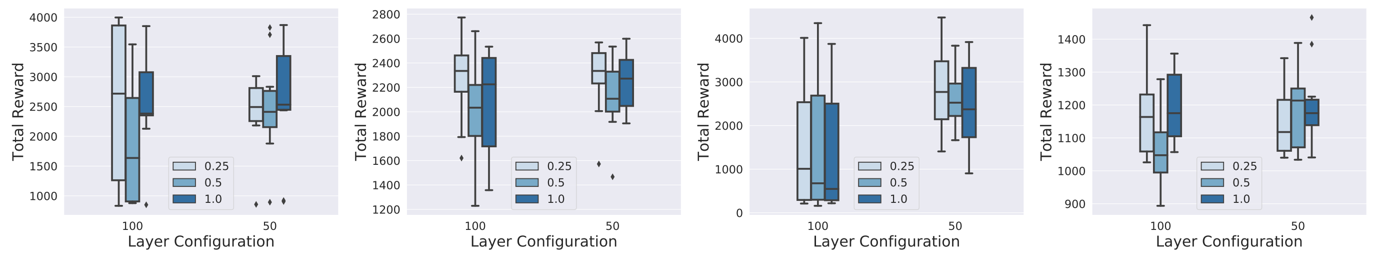
a) Novelty Search: First, we evaluate our method on a pure novelty search scenario by setting the reward pressure w to 0 and compare against classic method using k-nearest neighbors. As discussed in Sec III-C, NS methods require a domain-specific behavior characteristic (BC). For the pendulum tasks, BC is chosen as a trajectory of cart and pole positions. And for other locomotive tasks, it is chosen to be the 2-D trajectory of agent positions relative to the start position when the episode begins. Since agents might learn behaviors that move in the backward positions, it is necessary to align this BC with respect to the task of moving forward. For this reason, we clip the behavior space such that all behaviors with negative offsets (with respect to initial position) collapse to zero offsets. Fig 1 shows the learning curves of agents trained using only novelty gradients averaged over several runs. The important result is that our method of novelty-guided by behavior auto-encoders outperforms classic novelty methods in the tasks of Half-Cheetah, Hopper and Inverted Pendulum. In the task of Inverted Double Pendulum, classic novelty search performed slightly better, yet both the methods cannot solve the task while pursuing the novelty alone.

b) Novelty-Guided RL: Although novelty search methods are able to learn the necessary tasks skills without having access to reward functions, they still ignore some other aspects of reward functions such as energy efficiency, performance etc. To that end, they must be combined with reward gradients



(a) Half Cheetah (b) Inverted Pendulum (c) Inverted Double Pendulum (d) Hopper

Fig. 3: Effects of sparsity levels in pure novelty search methods. It is evident that sparse representations perform better than dense encodings in most settings.



(a) Half Cheetah (b) Inverted Pendulum (c) Inverted Double Pendulum (d) Hopper

Fig. 4: Effects of sparsity levels in novelty guided methods. It is evident that sparse representations perform better than dense encodings in most settings.

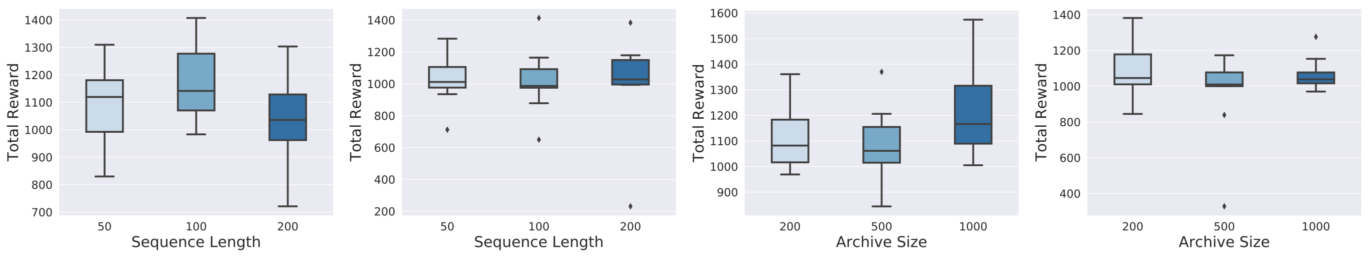
by adapting reward pressure accordingly based on learning progress as discussed in Sec III-D. To benchmark our methods, we consider the policy gradient method namely Evolution Strategies (ES) that considers only reward signals. For the adaptive methods combining with RL, we initially set the reward pressure to $w = 1.0$. The maximum stagnation steps is $k_{max} = 50$ and $\delta_w = 0.05$. Fig 2 shows the learning curves averaged over several runs. Our important results are summarized as (i) novelty guided methods which use both novelty and reward signals perform better than ES and speed up leaining in almost tasks (ii) Novelty guided by behavior models outperformed classic novelty search in three out of four tasks and performed equally well in the other tasks. (iii) The benefit of novelty-guided methods is clearly observed in difficult tasks such as half-cheetah, hopper etc. Also, it can be seen that scores of pure novelty search (Fig 1 (a), (d)) are higher than that of novelty-guided methods (Fig 2 (a), (d)) in some tasks. This could be because, novelty-guided methods also use reward signals, which penalize agents for its cost of motor actions, which might hinder the learning, whereas pure novelty search ignores these aspects, thereby not impeding the learning process.

A. *Sparsity Levels*

To understand the effect of enforced sparsity constraint, we considered behavior auto-encoders with different hidden layer configuration and sparsity levels $sparse$ of 0.25, 0.5 and 1.0. Note that the sparsity level of 1.0 corresponds to the classic auto-encoder in which all of the hidden units are used for reconstructing the given input. For this analysis, we consider the final performance of these variants over several runs. Fig 3 captures the effects of sparsity when using only the novelty gradients, with the help of a box plot showing the distribution of final performance. As it is observed, in most settings, the sparse encodings with levels of 0.25 and 0.5 perform better than the dense encoding of 1.0. Next, we performed the same analysis by considering the final performance when both novelty scores and rewards are used for learning. The plots in Fig 4 also suggests that the sparse encodings also performed better in most cases. In a nutshell, these results indicate that sparse representations are preferred over dense; however, this is another hyper-parameter which has to be tuned for the given task at hand.

B. *Ablation analysis*

a) Effect of Sequence Length: As discussed earlier, the chosen behavior characteristic is a sequence of 2D agent positions at specific intervals. For instance, in the task of Hopper, the sequence length is fixed to be 50. However, we would like to vary the sequence length and compare its influence on the final performance for both the novelty guided methods. We chose Hopper task as a testbed for this analysis as it is one of the difficult tasks. Fig 5 (a), (b) shows the box plot showing the performance using different sequence lengths. As seen in Fig 5 (a), the performance of neighborhood methods is affected by the sequence length; more importantly, it does



(a) Sequence Length: KNN (b) Sequence Length: AE (c) Archive Size: KNN (d) Archive Size: AE

Fig. 5: Ablation analysis: The plots (a),(b) shows the effect of sequence length on the performance for KNN and AE methods. Similarly, plots (c),(d) shows the effect of archive size on the performance for KNN and AE methods. It is observed that AE scales well to longer sequences and it requires only small buffer for learning

not scale well to longer sequence lengths. As seen in Fig 5 (b), for auto-encoders, the increased sequence length yields better results than shorter ones which show the scalability of function approximators using autoencoders.

b) Effect of archive size: The neighborhood models rely on a fixed-sized archive set using which the novelty scores are computed. Similarly, for autoencoders, a fixed-sized behavior buffer is used to sample behaviors to fit the model. To understand the influence of the size of the archive set and the behavior buffer, we performed an analysis on the Hopper task again. Fig 5 (c), (d) shows the box plot showing the performance using different sizes for the neighborhood models and autoencoders respectively. Clearly, the size influences the performance of neighborhood models, as seen in 5 (c). Larger the size, better the performance indicating that accurate novelty scores can be computed using a larger sized archive set. On the other hand, as it is observed in 5 (d), the size of the buffer does not influence the performance. In fact, even with a smaller buffer, the results are still competent. This shows that the autoencoders rely on encoding the behaviors in the weights of the network and rely less on stored behaviors in the buffer, therefore it can scale better compared to neighborhood models. Meaning, with further investigation and analysis, the behavior buffer can also be dropped entirely while the learning can be performed online using the sampled behaviors at each iteration. Regarding computation costs, the cost of computing novelty score is independent of buffer size K when using auto-encoders. In contrast, for the classic novelty search, it is $\mathcal{O}(K)$, which we have avoided with the auto-encoder approach. Yet, this scalability and generalizability come at the cost of additional training time of auto-encoders.

VI. Conclusion

In this paper, we addressed the limitations of novelty-search methods for reinforcement learning, which often suffer from lack of scalability or generalizability. We proposed a simple and scalable approach based on sparse behavior auto-encoders that can assign exploration bonuses to novel policies. Experimental results obtained for continuous control tasks suggest that our approach provides a viable alternative to novelty-search methods that classically rely on the notion of nearest neighbors among known policies.

Our work opens up several directions for further research. First of all, the use of behavioral characteristics instead of reward functions could also be an alternative for reinforcement learning settings where one has to specify rewards at each step. Here, it is fairly straight-forward to specify a behavior characteristic rather than specifying reward functions. Yet, modeling or learning of appropriate behavioral characteristics still is a largely unexplored research area. Here it seems auspicious to pursue the idea of informed reinforcement, that is, the idea of leveraging domain-knowledge in the design of the learning procedure.

Second, learning representations for sequences is an active area of research in several topics of machine learning such as natural language processing, speech recognition etc. Our work takes a first step towards applying such methods in the context of novelty-search methods. We hope that this motivates further work in the learning of efficient agent behavior representations using sequence to sequence architectures, which is devoid of fixed-length assumptions and is robust to temporal variations.

References

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI Safety. *arXiv preprint arXiv:1606.06565*, 2016.

[2] Kristopher De Asis, J. Fernando Hernandez-Garcia, G. Zacharias Holland, and Richard S. Sutton. Multi-step Reinforcement Learning: A Unifying Algorithm. *arXiv preprint arXiv:1703.01327*, 2017.

[3] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. *arXiv preprint arXiv:1606.01868*, 2016.

[4] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. *arXiv preprint arXiv:1712.06560*, 2017.

[5] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. *arXiv preprint arXiv:1802.06070*, 2018.

[6] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-policy Updates. In *Proc. Int. Conf. Robotics and Automation*, 2017.

[7] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, Martin Riedmiller, et al. Emergence of Locomotion Behaviours in Rich Environments. *arXiv preprint arXiv:1707.02286*, 2017.

[8] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in NIPS*, 2016.

[9] Joel Lehman and Kenneth O Stanley. Abandoning Objectives: Evolution through the Search for Novelty Alone. *Evolutionary computation*, 19(2), 2011.

[10] Joel Lehman and Kenneth O Stanley. Evolving a diversity of Virtual Creatures through Novelty Search and Local Competition. In *Proc. Int. Conf on Genetic and Evolutionary Computation*, 2011.

[11] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. Learning Behaviors via Human-delivered Discrete Feedback: Modeling Implicit Feedback Strategies to speed up Learning. *In Proc. Conf. on Aut. Agents and Multi. Systems*, 30(1), 2016.

[12] Alireza Makhzani and Brendan Frey. K-sparse Autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

[13] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to Navigate in Complex Environments. *arXiv preprint arXiv:1611.03673*, 2016.

[14] 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), 2015.

[15] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In *Proc. Int. Conf. Machine Learning*, 1999.

[16] Andrew Y Ng, Stuart J Russell, et al. Algorithms for Inverse Reinforcement Learning. In *Proc. Int. Conf. on Machine Learning*, 2000.

[17] Georg Ostrovski, Marc G. Bellemare, Aaron van den Oord, and Remi Munos. Count-Based Exploration with Neural Density Models, 2017.

[18] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-Driven Exploration by Self-supervised Prediction. In *Proc. Int. Conf. Machine Learning*, 2017.

[19] Pathak, Deepak and Gandhi, Dhiraj and Gupta, Abhinav. Self-Supervised Exploration via Disagreement. *arXiv preprint arXiv:1906.04161*, 2019.

[20] Bei Peng, James MacGlashan, Robert Loftin, Michael L Littman, David L Roberts, and Matthew E Taylor. A Need for Speed: Adapting Agent Action Speed to Improve Task Learning from Non-Expert Humans. In *Proc. Int. Conf. on Aut. Agents & Multi. Systems*, 2016.

[21] Rajkumar Ramamurthy, Christian Bauckhage, Rafet Sifa, Jannis Schücker, and Stefan Wrobel. Leveraging Domain Knowledge for Reinforcement Learning Using MMC Architectures. In *Proc. of Int. Conf. on. Artificial Neural Networks*, 2019.

[22] Rajkumar Ramamurthy, Christian Bauckhage, Rafet Sifa, and Stefan Wrobel. Policy Learning Using SPSA. In *Proc. Int. Conf. on Artificial Neural Networks*, 2018.

[23] Jette Randløv and Preben Alstrøm. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *Proc. Int. Conf. on Machine Learning*, 1998.

[24] I Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Thesis, Technical University of Berlin, Department of Process Engineering, 1971.

[25] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*. 1978.

[26] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864*, 2017.

[27] Jürgen Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2, 2010.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proc. Int. Conf. Machine Learning*, 2015.

[29] Craig Sherstan, Brendan Bennett, Kenny Young, Dylan R Ashley, Adam White, Martha White, and Richard S Sutton. Directly Estimating the Variance of the λ-return using Temporal-Difference Methods. *arXiv preprint arXiv:1801.08287*, 2018.

[30] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proc. Int. Conf on Machine Learning*, 2014.

[31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the Game of Go without Human Knowledge. *Nature*, 550(7676), 2017.

[32] Bradly C. Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models, 2015.

[33] Halit Bener Suay, Tim Brys, Matthew E Taylor, and Sonia Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *Proc. Int. Conf. on Aut. Agents & Multi. Systems*, 2016.

[34] 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 *Proc. Advances in NIPS*, 2017.

[35] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-Driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *Proc. Int. Conf. Robotics and Automation*, 2017.