

Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents

Edoardo Conti* Vashisht Madhavan* Felipe Petroski Such Joel Lehman Kenneth O. Stanley Jeff Clune

Uber AI Labs

{edoardo, vashisht, jeffclune}@uber.com

Abstract

Evolution strategies (ES) are a family of black-box optimization algorithms able to train deep neural networks roughly as well as Q-learning and policy gradient methods on challenging deep reinforcement learning (RL) problems, but are much faster (e.g. hours vs. days) because they parallelize better. However, many RL problems require directed exploration because they have reward functions that are sparse or deceptive (i.e. contain local optima), and it is not known how to encourage such exploration with ES. Here we show that algorithms that have been invented to promote directed exploration in small-scale evolved neural networks via populations of exploring agents, specifically novelty search (NS) and quality diversity (QD) algorithms, can be hybridized with ES to improve its performance on sparse or deceptive deep RL tasks, while retaining scalability. Our experiments confirm that the resultant new algorithms, NS-ES and a version of QD we call NSR-ES, avoid local optima encountered by ES to achieve higher performance on tasks ranging from playing Atari to simulated robots learning to walk around a deceptive trap. This paper thus introduces a family of fast, scalable algorithms for reinforcement learning that are capable of directed exploration. It also adds this new family of exploration algorithms to the RL toolbox and raises the interesting possibility that analogous algorithms with multiple simultaneous paths of exploration might also combine well with existing RL algorithms outside ES.

1. Introduction

In RL, an agent tries to learn to perform a sequence of actions in an environment that maximizes some notion of cumulative reward (Sutton & Barto, 1998). However, reward functions are often *deceptive*, and solely optimizing for reward without some mechanism to encourage intelligent exploration can lead to getting stuck in local optima and the agent failing to properly learn (Liepins & Vose, 1990; Lehman & Stanley, 2011a; Sutton & Barto, 1998). Unlike in supervised learning with deep neural networks (DNNs), wherein local optima are not thought to be a problem (Kawaguchi, 2016; Dauphin et al., 2014), the training data in RL is determined by the actions an agent takes. If the agent greedily takes actions that maximize reward, the training data for the algorithm will be limited and it may not discover alternate strategies with larger payoffs (i.e. it can get stuck in local optima) (Liepins & Vose, 1990; Lehman & Stanley, 2011a; Sutton & Barto, 1998). Sparse reward signals can also be a problem for algorithms that only maximize reward, because at times there may be no reward gradient to follow. The possibility of deceptiveness and or sparsity in the reward signal motivates the need for efficient and *directed* exploration, in which an agent is motivated to visit unexplored states in order to learn to accumulate higher rewards. Although deep RL algorithms have performed amazing feats in recent years (Mnih et al., 2015; 2016; Schulman et al., 2015), they have mostly done so despite relying on simple, *undirected* (aka dithering) exploration strategies, in which an agent hopes to explore new areas of its environment by taking random actions (e.g. epsilon-greedy exploration) (Sutton & Barto, 1998).

A number of methods have been proposed to promote directed exploration in RL (Schmidhuber, 2010; Oudeyer & Kaplan, 2009), including recent methods that handle high-dimensional state spaces with deep neural networks. A common idea is to encourage an agent to visit states it has rarely or never visited (or take novel actions in those states). Methods proposed to track how often states or state-action pairs have been visited include (1) approximating state visitation counts based on either auto-encoded latent codes of

*Both authors contributed equally to this work.

states (Tang et al., 2017) or pseudo-counts from state-space density models (Bellemare et al., 2016; Ostrovski et al., 2017), (2) learning a dynamics model that predicts future states (assuming predictions will be worse for rarely visited states/state-action pairs) (Stadie et al., 2015; Houthoofd et al., 2016; Pathak et al., 2017), and (3) methods based on compression (novel states should be harder to compress) (Schmidhuber, 2010).

Those methods all count each state separately. A different approach is to hand-design (or learn) an abstract, holistic description of the overall behavior of an agent throughout its lifetime, and then encourage the agent to exhibit different behaviors from those it has previously performed. That is the approach of novelty search (Lehman & Stanley, 2011a) and quality diversity algorithms (Cully et al., 2015; Mouret & Clune, 2015; Pugh et al., 2016), which are described in detail below. Here we hybridize such algorithms with ES and demonstrate that they do improve exploration on hard deep RL problems, and do so without sacrificing the speed/scalability benefits of ES.

There is another interesting difference between the other exploration methods mentioned previously and the NS/QD family of algorithms. We do not investigate the benefits of this difference experimentally in this paper, but they are one reason we are interested in NS/QD as an exploration method for RL. One commonality among the previous methods is that the exploration is performed by a *single* agent, a choice that has interesting consequences for learning. To illustrate these consequences, we borrow an example from Stanton & Clune (2016). Imagine a cross-shaped maze (SI Sec. 6.2) where to go in each cardinal direction an agent must master a different skill (e.g. going north requires learning to swim, west requires climbing mountains, east requires walking on sand, and south requires walking on ice). Assume rewards may or may not exist at the end of each corridor, so all corridors need to be explored. A single agent has two extreme options, either a depth-first search that serially learns to go to the end of each corridor, or a breadth-first search that goes a bit further in one direction, then comes back to the center and goes a bit further in another direction, etc. Either way, to get to the end of each hallway, the agent will have to at least have traversed each hallway once and thus will have to learn all four sets of skills. With the breadth-first option, all four skillsets must be mastered, but a much longer total distance is traveled.

In both cases, another problem arises because, despite recent progress (Kirkpatrick et al., 2017; Velez & Clune, 2017), neural networks still suffer from catastrophic forgetting, meaning that as they learn new skills they rapidly lose the ability to perform previously learned ones (French, 1999). Due to catastrophic forgetting, at the end of learning there will be an agent specialized in one of the skills

(e.g. swimming), but all of the other skills will have been lost. Furthermore, if any amount of the breadth-first search strategy is employed, exploring each branch a bit further will require relearning that skill mostly from scratch each iteration, significantly slowing exploration. Even if catastrophic forgetting could be solved, there may be limits on the cognitive capacity of single agents (as occurs in humans), preventing one agent from mastering all possible skills.

A different approach is to explore with a *population* of agents. In that case, separate agents could become experts in the separate tasks required to explore in each direction. That may speed learning because each agent can, in parallel, learn only the skills required for its corridor. Additionally, at the end of exploration a specialist will exist with each distinct skill (versus only one skill remaining in the single-agent case). The resulting population of specialists, each with a different skill or way of solving a problem, can then be harnessed by other machine learning algorithms that efficiently search through the repertoire of specialists to find the skill or behavior needed in a particular situation (Cully et al., 2015; Cully & Mouret, 2013). The skills of each specialist (in any combination or number) could also then be combined in to a single generalist via policy distillation (Rusu et al., 2015). A further benefit of the population-based approach is, when combining exploration with some notion of quality (e.g. maximizing reward), a population can try out many different strategies/directions and, once one or a few are found to be promising, the algorithm can reallocate resources to pursue the most promising directions. The point is not that population-based exploration methods are better or worse than single-agent exploration methods (when holding computation constant), but instead that they are a different option with different capabilities, pros, and cons, and are thus worth investigating (Stanton & Clune, 2016). Supporting this view, recent work has demonstrated the benefits of populations for deep learning (Jaderberg et al., 2017; Miikkulainen et al., 2017).

Novelty search and quality diversity have shown promise with smaller neural networks on problems with low-dimensional input and output spaces (Lehman & Stanley, 2011c; Cully et al., 2015; Mouret & Clune, 2015; Pugh et al., 2016; Velez & Clune, 2014; Huizinga et al., 2016). In this paper, for the first time, we study how these two types of algorithms can be hybridized with ES in order to scale them to deep neural networks and thus tackle hard, high-dimensional deep reinforcement learning problems. We first study an algorithm called *novelty search* (NS) (Lehman & Stanley, 2011a), which performs exploration *only* (ignoring the reward function) to find a set of novel solutions. We then investigate an algorithm that balances exploration and exploitation, specifically a novel instance of a *quality diversity* (QD) algorithm, which seeks to produce

a set of solutions that are both novel and high-performing (Lehman & Stanley, 2011c; Cully et al., 2015; Mouret & Clune, 2015; Pugh et al., 2016). Both NS and QD are explained in detail in Sec. 3.

ES directly searches in the parameter space of a neural network to find an effective policy. A team from OpenAI recently showed that ES can achieve competitive performance on many reinforcement learning (RL) tasks while offering some unique benefits over traditional gradient-based RL methods (Salimans et al., 2017). Most notably, ES is highly parallelizable, which enables near linear speedups in runtime as a function of CPU/GPU workers. For example, with hundreds of parallel CPUs, ES was able to achieve roughly the same performance on Atari games with the same DNN architecture in 1 hour as A3C did in 24 hours (Salimans et al., 2017). In this paper, we investigate adding NS and QD to ES only; in future work, we will investigate how they might be hybridized with Q-learning and policy gradient methods. We start with ES because (1) its fast wall-clock time allows rapid experimental iteration, (2) NS and QD were originally developed as neuroevolution methods, making it natural to try them first with ES, which is also an evolutionary algorithm, (3) it is more straightforward to integrate population-based exploration with ES than with Q-learning, and (4) our team is most familiar with the NS and QD family of exploration algorithms than the methods that treat each state separately.

Here we test whether encouraging novelty via NS and QD improves the performance of ES on sparse and/or deceptive control tasks. Our experiments confirm that NS-ES and a simple version of QD-ES (called NSR-ES) avoid local optima encountered by ES and achieve higher performance on tasks ranging from simulated robots learning to walk around a deceptive trap to the high-dimensional pixel-to-action task of playing Atari games. Our results add these new families of exploration algorithms to the RL toolbox, opening up avenues for studying how they can best be combined with RL algorithms, whether ES or others, and comparing these new types of population-based exploration methods to traditional ones.

2. Background

2.1. Evolution Strategies

Evolution strategies (ES) are a class of black box optimization algorithms inspired by natural evolution (Rechenberg, 1978): At every iteration (generation), a population of parameter vectors (genomes) is perturbed (mutated) and, optionally, recombined (merged) via crossover. The reward (fitness) of each resultant offspring is then evaluated according to some objective function. Some form of selection then ensures that individuals with higher reward tend to

produce the individuals in the next generation, and the cycle repeats. Many algorithms in the ES class differ in their representation of the population and methods of recombination; the algorithms subsequently referred to in this work belong to the class of Natural Evolution Strategies (NES) (Wierstra et al., 2008; Sehne et al., 2010). NES represents the population as a distribution of parameter vectors θ characterized by parameters ϕ : $p_\phi(\theta)$. Under a fitness function, $f(\theta)$, NES seeks to maximize the average fitness of the population, $\mathbb{E}_{\theta \sim p_\phi}[f(\theta)]$, by optimizing ϕ with stochastic gradient ascent.

Recent work from OpenAI outlines a version of NES applied to standard RL benchmark problems (Salimans et al., 2017). We will refer to this variant simply as ES going forward. In their work, a fitness function $f(\theta)$ represents the stochastic reward experienced over a full episode of agent interaction, where θ is the parameters of a policy π_θ . The population distribution p_{ϕ_t} is an isotropic multivariate Gaussian with mean θ_t , the parameter vector at iteration t , and covariance $\sigma^2 I$ (i.e. $\mathcal{N}(\theta_t, \sigma^2 I)$). From the distribution, parameters $\theta_t^i \sim \mathcal{N}(\theta_t, \sigma^2 I)$ are sampled and their corresponding policies $\pi_{\theta_t^i}$ are evaluated to obtain a reward $f(\theta_t^i)$. In a manner similar to REINFORCE (Williams, 1992), the approximate gradient of expected reward with respect to θ_t can be found with the gradient estimator:

$$\nabla_{\phi} \mathbb{E}_{\theta \sim \phi}[f(\theta)] \approx \frac{1}{n} \sum_{i=1}^n f(\theta_t^i) \nabla_{\phi} \log p_{\phi}(\theta_t^i)$$

where n is the number of samples evaluated per generation. Intuitively, NES samples parameters in the neighborhood of θ_t and determines the direction in which θ_t should move to improve expected reward. Instead of a baseline, NES relies on a large number of samples n to reduce the variance of the gradient estimator. Generally, NES evolves both the mean and covariance of the population distribution, but for the sake of fair comparison with Salimans et al. (2017) we consider only static covariance distributions, meaning σ is fixed throughout training.

To simplify the optimization process, Salimans et al. (2017) reformulates sampling from the population distribution as applying additive Gaussian noise to the current parameter vector: $\theta_t^i = \theta_t + \sigma \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, I)$. The gradient estimate can then be found by taking a sum of sampled parameter perturbations weighted by their reward:

$$\nabla_{\theta_t} \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)}[f(\theta_t + \sigma \epsilon)] \approx \frac{1}{n\sigma} \sum_{i=1}^n f(\theta_t^i) \epsilon_i$$

To ensure that the scale of reward between domains does not bias the optimization process, we follow the approach of Salimans et al. (2017) and rank-normalize $f(\theta_t^i)$ before taking the weighted sum. Overall, this NES variant by Sal-

imans et al. (2017) exhibits performance on par with contemporary, gradient-based algorithms when applied to difficult RL domains, including simulated robot locomotion and Atari 2600 (Bellemare et al., 2013) environments.

2.2. Novelty Search (NS)

Optimizing for reward only can often lead an agent to local optima. NS, however, avoids deception in the reward signal by ignoring reward altogether. Inspired by nature’s drive towards diversity, NS encourages policies to engage in notably different behaviors than those previously seen. The algorithm encourages different behaviors by computing the *novelty* of the current policy with respect to previously generated policies and then encourages the population distribution to move towards areas of parameter space with high novelty. NS outperforms reward-based methods in maze and biped walking domains, which possess deceptive reward signals that attract agents to local optima (Lehman & Stanley, 2011a). In this work, we investigate the efficacy of NS at the scale of DNNs by combining it with ES. In a companion paper, we investigate NS at DNNs scales evolved with a simple GA instead of via ES (Petroski Such et al., 2017).

In NS, a policy π is assigned a domain-dependent *behavior characterization* $b(\pi)$ that describes its behavior. For example, in the case of a humanoid locomotion problem, $b(\pi)$ may be as simple as a two-dimensional vector containing the humanoid’s final $\{x, y\}$ location. Throughout training, every π_θ evaluated adds a behavior characterization $b(\pi_\theta)$ to an archive set A with some probability. A particular policy’s novelty $N(b(\pi_\theta), A)$ is then computed by selecting the k -nearest neighbors of $b(\pi_\theta)$ from A and computing the average distance between them:

$$N(\theta, A) = N(b(\pi_\theta), A) = \frac{1}{|S|} \sum_{j \in S} \|b(\pi_\theta) - b(\pi_j)\|_2$$

$$S = kNN(b(\pi_\theta), A)$$

$$= \{b(\pi_1), b(\pi_2), \dots, b(\pi_k)\}$$

Above, the distance between behavior characterizations is calculated with an $L2$ -norm, but an arbitrary distance function can be substituted.

Previously, NS has been implemented with a genetic algorithm (Lehman & Stanley, 2011a). The next section explains how NS can now be combined with ES, to leverage the advantages of both algorithms.

3. Methods

3.1. NS-ES

We use the ES optimization framework, described in Sec. 2.1, to compute and follow the gradient of expected

novelty with respect to θ_t . Given an archive A and sampled parameters $\theta_t^i = \theta_t + \sigma \epsilon_i$, the gradient estimate can be computed:

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

The gradient estimate obtained tells us how to change the current policy’s parameters θ_t to increase the average novelty of our parameter distribution. We condition the gradient estimate on A , as the archive is fixed at the beginning of a given iteration and updated only at the end. We add only the behavior characterization corresponding to each θ_t , as adding those for each sample θ_t^i would inflate the archive and slow the nearest-neighbors computation. As more behavior characterizations are added to A , the novelty landscape changes, resulting in commonly occurring behaviors becoming “boring”. Optimizing for expected novelty leads to policies that move towards unexplored areas of behavior space.

NS-ES could operate with a single agent that is rewarded for acting differently than its ancestors. However, to encourage additional diversity and get the benefits of population-based exploration described in Sec. 1, we can instead create a population of M agents, which we will refer to as the *meta-population*. Each agent, characterized by a unique θ^m , is rewarded for being different from all prior agents in the archive (ancestors, other agents, and the ancestors of other agents). In this paper, we have multiple agents in the meta-population of our experiments (i.e. $M > 1$) because we thought it would help, but we did not conduct a thorough analysis on how varying this hyperparameter affects performance on different domains. We hypothesize that the selection of M is domain dependent and that identifying which domains favor which regime is a fruitful area for future research.

We initialize M random parameter vectors and at every iteration select one to update. For our experiments, we probabilistically select which θ^m to advance from a discrete probability distribution as a function of θ^m ’s novelty. Specifically, at every iteration, for a set of agent parameter vectors $\Pi = \{\theta^1, \theta^2, \dots, \theta^M\}$, we calculate each θ^m ’s probability of being selected $P(\theta^m)$ as its novelty normalized by the sum of novelty across all policies:

$$P(\theta^m) = \frac{N(\theta^m, A)}{\sum_{j=1}^M N(\theta^j, A)} \quad (1)$$

Having multiple, separate agents represented as independent Gaussians is a simple choice for the *meta-population* distribution (i.e. how the meta-population distribution is represented). In future work, more complex sampling distributions that represent the multi-modal nature of meta-population parameter vectors could be tried.

After selecting a certain individual m from the meta-population, we compute the gradient of expected novelty with respect to m 's current parameter vector, θ_t^m , and perform an update step accordingly:

$$\theta_{t+1}^m \leftarrow \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^n N(\theta_t^{i,m}, A) \epsilon_i$$

Where n is the number of sampled perturbations to θ_t^m , α is the stepsize, and $\theta_t^{i,m} = \theta_t^m + \sigma \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, I)$. Once the current parameter vector is updated, $b(\pi_{\theta_{t+1}^m})$ is computed and added to the shared archive A with probability 1. The whole process is repeated for a pre-specified number of iterations, as there is no true convergence point of NS. In this work, the algorithm simply returns the highest-performing parameter vector found. Algorithm 1 in SI Sec. 6.3 outlines a simple, parallel implementation of NS-ES. It is important to note that the addition of the archive and the replacement of the fitness function with novelty does not damage the scalability of the ES optimization procedure (SI Sec. 6.4).

3.2. A QD-ES algorithm: NSR-ES

NS-ES alone can enable agents to avoid deceptive local optima in the reward function. Reward signals, however, are still very informative and discarding them completely may cause performance to suffer. Consequently, we train a variant of NS-ES, which we call NSR-ES, that combines the reward ("fitness") and novelty calculated for a given set of policy parameters θ . Similar to NS-ES and ES, NSR-ES operates on entire episodes and can thus evaluate reward and novelty simultaneously for any sampled parameter vector: $\theta_t^{i,m} = \theta_t^m + \epsilon_i$. Specifically, we compute $f(\theta_t^{i,m})$ and $N(\theta_t^{i,m}, A)$, average the two values, and set the average as the weight for the corresponding ϵ_i . The averaging process is integrated into the parameter update rule as

$$\theta_{t+1}^m \leftarrow \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^n \frac{f(\theta_t^{i,m}) + N(\theta_t^{i,m}, A)}{2} \epsilon_i$$

Intuitively, the algorithm hill-climbs in parameter-space towards policies that both exhibit novel behaviors and achieve high rewards. Often, however, the scales of $f(\theta)$ and $N(\theta, A)$ differ. To combine the two signals effectively, we rank-normalize $f(\theta_t^{i,m})$ and $N(\theta_t^{i,m}, A)$ independently before computing the average.

Averaging $f(\theta_t^{i,m})$ and $N(\theta_t^{i,m}, A)$ is a relatively simple way of encouraging both quality and diversity. More intricate methods of combining the two desiderata, including simple weighted averaging, are left for future work.

Here we do not take advantage of the entire set of diverse, high-performing individuals, but instead the algorithm simply returns the best parameter vector found. This work is

thus a preliminary step in applying QD algorithms to common deep RL benchmarks. Further research may investigate more sophisticated QD methods. SI Sec. 6.3 provides pseudocode for NSR-ES. In the future, we plan to release source code and hyperparameter configurations for all of our experiments.

4. Experiments

4.1. Simulated Humanoid Locomotion problem

We first tested our implementation of NS-ES and NSR-ES on the problem of having a simulated humanoid learn to walk. We chose this problem because it is a challenging continuous control deep reinforcement learning benchmark where most would presume a reward function is necessary to solve the problem. With NS-ES, we test whether searching through novelty alone can find solutions to the problem. A similar result has been shown for much smaller neural networks (~ 50 -100 parameters) on a more simple simulated biped (Lehman & Stanley, 2011c), but here we test whether NS-ES can enable the same results at the scale of deep neural networks. NSR-ES experiments test the effectiveness of combining exploration and reward pressures on this difficult continuous control problem.

Specifically, the domain is the MuJoCo Humanoid-v1 environment in OpenAI Gym (Brockman et al., 2016). In it, a humanoid robot receives a scalar reward composed of four components per timestep. The robot gets positive reward for standing and velocity in the positive x direction, and negative reward for ground impact energy and energy expended. These four components are summed across every timestep in an episode to get the total reward. Following the neural network architecture outlined by Salimans et al. (2017), the neural network is a multilayer perceptron with two hidden layers containing 256 neurons each, resulting in a network with 166.7K parameters. While small especially in the number of layers compared to many deep RL architectures, this network is still orders of magnitude larger than what NS has been tried with before. We also test NS with a 1M+ parameter network (Sec. 4.2). The input to the network is the observation space from the environment, which is a vector $\in \mathbb{R}^{376}$ representing the state of the humanoid (e.g. joint angles, velocities) and the output of the network is a vector of motor commands $\in \mathbb{R}^{17}$ (Brockman et al., 2016). Complete training details and hyperparameters are in SI Sec. 6.6.

The first experiment compares ES, NS-ES, and NSR-ES on a slightly modified version of OpenAI Gym's Humanoid-v1 environment. Because the heart of this challenge is to learn to walk efficiently, not to walk in a particular direction, we modified the environment reward to be indifferent to the direction the humanoid traveled. Specifically,

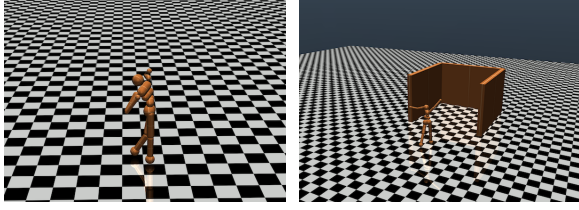


Figure 1. The Humanoid Locomotion problem. In the default Humanoid-v1 Environment, a simulated robot learns to walk in an open environment with no walls (left). We created the *Humanoid Locomotion with Deceptive Trap* problem (right) to test algorithms in an RL domain with a clear local optimum. The agent receives more reward the further it walks in the direction of the trap (the small enclosure). Algorithms that do not sufficiently explore may walk into the trap and get stuck, while algorithms that sufficiently encourage exploration over exploitation can avoid the trap.

the modified reward function is isotropic (i.e. the velocity component of reward is based on distance traveled from the origin as opposed to distance traveled in the positive x direction).

As described in section 2.2, novelty search requires a domain-specific behavior characterization of each policy $b(\pi_{\theta_i})$. For the Humanoid Locomotion problem the BC is the agent’s final $\{x, y\}$ location, as it was in [Lehman & Stanley \(2011c\)](#). In addition to a behavioral characterization, NS also requires a distance function between two behavioral characterizations. Following Lehman & Stanley 2011 ([Lehman & Stanley, 2011c](#)), the distance function is the square of the Euclidean distance between two BCs:

$$\text{dist}(b(\pi_{\theta_i}), b(\pi_{\theta_j})) = (\|b_t(\pi_{\theta_i}) - b_t(\pi_{\theta_j})\|_2)^2$$

The first result is that ES obtains a higher final reward than NS-ES ($p < 0.05$) and NSR-ES ($p < 0.05$; these and all future p values are via a Mann-Whitney U test). Its performance gap is even more pronounced for smaller amounts of computation (Fig. 2). However, many will be surprised that NS-ES is still able to consistently solve the problem despite ignoring the environment’s multi-part reward function. While the BC is *aligned* ([Pugh et al., 2015](#)) with the problem in that reaching new $\{x, y\}$ positions tends to also encourage walking, there are many parts of the reward function for which the BC does not obviously help (e.g. energy-efficient, low-impact locomotion).

We hypothesize that with a sophisticated BC that encourages diversity in all of the behaviors the multi-part reward function cares about, there would be no performance gap. However, such a BC may be difficult to construct and would likely further exaggerate the amount of computation required for NS to match ES. NSR-ES demonstrates faster learning than NS-ES due to the addition of reward pressure, but ultimately results in similar final performance after 600

generations ($p > .05$) (Fig. 2).

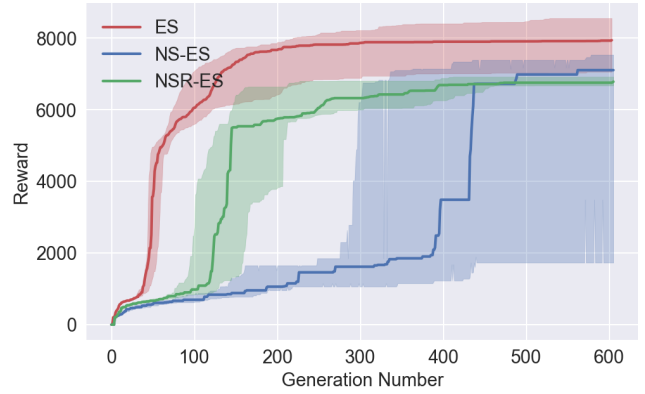


Figure 2. Although slower than ES, NS-ES and NSR-ES arrive at competitive solutions on average. While ES outperforms NS-ES and NSR-ES, it is interesting how well NS-ES does given that it ignores the reward function. The addition of the reward pressure helps NSR-ES perform much better earlier than NS-ES and to do so with lower variance. Overall all three algorithms learn competitive solutions to the problem in that they learn to walk quickly. Here and in similar figures below, the median reward (of the best seen policy so far) per generation across 10 runs is plotted as the bold line with 95% bootstrapped confidence intervals of the median (shaded). Policy performance is measured as average performance over ~ 30 stochastic evaluations.

The Humanoid Locomotion problem does not appear to be a deceptive problem, at least for ES. To test whether NS-ES and NSR-ES specifically help with deception, we also compare ES to these algorithms on a variant of this environment we created that adds a deceptive trap (a local optimum) that must be avoided for maximum performance (Fig. 1, right). In this new environment, a small three-sided enclosure is placed at a short distance in front of the starting position of the humanoid and the reward function is simply distance traveled in the positive x direction.

Fig. 4 and Table 6.7 show the reward received by each algorithm, and Fig. 3 shows how the algorithms differ qualitatively during search on this problem. In every run, ES gets stuck in the local optimum due to following reward into the deceptive trap. NS-ES is able to avoid the local optimum as it ignores reward completely and instead seeks to thoroughly explore the environment, but doing so also means it makes slow progress according to the reward function. NSR-ES demonstrates superior performance to NS-ES ($p < 0.01$) and ES ($p < 0.01$) as it benefits from both optimizing for reward and escaping the trap via the pressure for novelty.

Fig. 3 also shows the benefit of maintaining a meta-population ($M = 5$) in the NS-ES and NSR-ES algorithms. Some lineages get stuck in the deceptive trap, incentivizing other policies to explore around the trap. At that point, NS-

ES and NSR-ES begin to allocate more computational resources to this newly discovered, more promising strategy via the probabilistic selection method outlined in Sec. 3.1. Both the novelty pressure and having a meta-population thus appear to be useful, but in future work we look to disambiguate the relative contribution made by each.

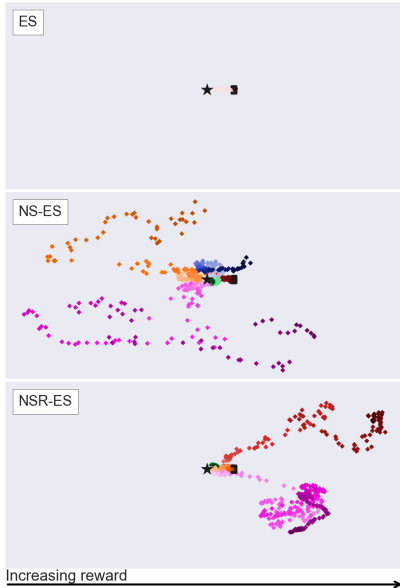


Figure 3. ES gets stuck in the deceptive local optimum while NS-ES & NSR-ES explore to find better solutions. An overhead view of a representative run is shown for each algorithm on the Humanoid Locomotion with Deceptive Trap problem. The black star represents the humanoid’s starting point. Each diamond represents the final location of a generation’s mean policy, i.e. $\pi(\theta_t)$, with darker shading for later generations. For NS-ES & NSR-ES plots, each of the $M = 5$ agents in the meta-population and its descendants are represented by different colors. With ES, the humanoid walks into the deceptive trap and never learns to navigate around it. NS-ES explores the solution space much more than ES and achieves a higher reward, but wastes significant computation exploring in low and negative reward areas to the left of the origin. NSR-ES has the best performance out of all 3 algorithms (Fig. 4). It generally walks in the direction indicated by the reward function, including walking into the trap, but its exploration pressure helps it discover other, better solutions that involve walking around the trap. Similar plots for all 10 runs of each algorithm are provided in SI. 6.9.

4.2. Atari

We also tested NS-ES and NSR-ES on numerous games from the Atari 2600 environment in OpenAI Gym (Brockman et al., 2016). Atari games serve as an informative benchmark due to their high-dimensional pixel input and complex control dynamics; each game also requires different levels of exploration to solve. To demonstrate the effectiveness of NS-ES and NSR-ES for local optima avoidance

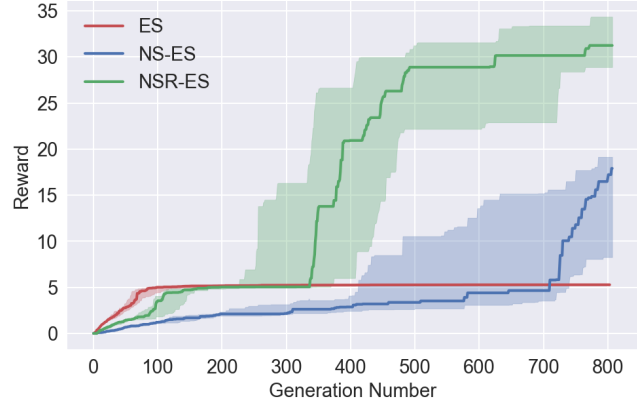


Figure 4. NSR-ES and NS-ES outperform ES on the Humanoid Locomotion with Deceptive Trap problem. ES walks into the trap and, because it has no exploration pressure, gets trapped there indefinitely. NS-ES outperforms ES as it avoids the local optimum, but requires significant computation to do so because it completely ignores the reward. NSR-ES shows the best performance, demonstrating the value of rewarding both exploration and performance.

and directed exploration, we tested on 12 different games with varying levels of complexity, as defined by the taxonomy in (Bellemare et al., 2016). Primarily, we focused on games in which, during preliminary experiments, we observed that our implementation of ES prematurely converges to local optima (Seaquest, Q*Bert, Freeway, Frostbite, and Beam Rider). However, we also included a few other games where ES did not converge to local optima to understand the performance of our algorithm in less-deceptive domains (Alien, Amidar, Bank Heist, Breakout, Gravitar, Zaxxon, and Montezuma’s Revenge). Since we are uncertain as to whether other papers report the average reward of the best single policy found by an algorithm in *any run*, or the median reward across r independent runs of the best policy found in each run, we report both (see Table 1 and Table 2)

As in Mnih et al. (2016), data preprocessing follows Mnih et al. (2015) and the network architecture is from Mnih et al. (2013). Each algorithm is evaluated over 5 separate runs. In this domain NS-ES and NSR-ES have three meta-population agents (i.e. $M = 3$), as each algorithm trained for fewer generations than in the Humanoid Locomotion task. We lowered M because the Atari network is much larger and thus each generation is more computationally expensive. A lower M enables more generations to occur in training.

For the behavior characterization, we follow an idea from Naddaf (2010) and concatenate Atari game RAM states for each timestep in an episode. RAM states in Atari 2600 games are integer-valued vectors of length 128 in the range

[0, 255] that describe all the state variables in a game (e.g. the location of the agent and enemies). Ultimately, we want to automatically learn behavior characterizations directly from pixels. A plethora of recent research suggests that is a viable approach (Lange & Riedmiller, 2010; Kingma & Welling, 2013; Bellemare et al., 2016). For example, low-dimensional, latent representations of the state space could be extracted from auto-encoders (Tang et al., 2017; van den Oord et al., 2016) or networks trained to predict future states (Pathak et al., 2017; Stadie et al., 2015). In this work, however, we focus on learning with a pre-defined, informative behavior characterization and leave the task of jointly learning a policy and latent representation of states for future work. In effect, basing novelty on RAM states provides a confirmation of what is possible in principle with a sufficiently informed behavior characterization. We also emphasize that, while during training NS-ES and NSR-ES use RAM states to guide novelty search, the policy itself, π_{θ_t} , operates only on image input and can be evaluated without any RAM state information. The distance between behavior characterizations is the sum of L2-distances at each timestep k :

$$\text{dist}(b(\pi_{\theta_i}), b(\pi_{\theta_j})) = \sum_{k=1}^K ||(b_t(\pi_{\theta_i})) - b_t(\pi_{\theta_j}))||_2$$

Table 1 and Table 2 compare the performance of the algorithms. While the novelty pressure in NS-ES does help it avoid local optima in some cases (discussed below), optimizing for novelty only does not result in higher reward in most games (although it does in some). However, it is surprising how well NS-ES does in many tasks given that it is not explicitly attempting to increase reward.

Because NSR-ES combines exploration with reward maximization, it is able to avoid local optima encountered by ES while also learning to play the game well. In each of the 5 games in which we observed ES converge to premature local optima (i.e. Seaquest, Q*Bert, Freeway, Beam Rider, Frostbite), NSR-ES achieves a higher median reward. NS-ES also tends to outperform ES in these games. In the other games, ES does not benefit from adding an exploration pressure and NSR-ES performs worse. It is expected that if there are no local optima and reward maximization is sufficient to perform well, the extra cost of encouraging exploration will hurt performance. We hypothesize that is what is occurring with these games. We note that all conclusions are premature, however, as we did not gather large enough sample sizes in the Atari domain to test whether these performance differences between algorithms on each game are statistically significant.

In the game Seaquest, the avoidance of local optima is particularly evident, as highlighted in Fig. 5. ES performance flatlines early at a median reward of 960, which

corresponds to a behavior of the agent descending to the bottom, shooting fish, and never coming up for air. This strategy represents a classic local optima, as coming up for air requires temporarily foregoing reward, but enables far higher rewards to be earned in the long run. NS-ES learns to come up for air in all 5 runs and achieves a significantly higher median reward of 1044.5 ($p < 0.05$). NSR-ES also avoids this local optima, but its additional reward signal helps it play the game better (e.g. it is better at shooting enemies), resulting in a much higher median reward of 2329.7 ($p < 0.01$). Our experimental results for ES on Seaquest differ from those of the blog post associated with Salimans et al. (2017), as it report agents learning to come up for air (our hyperparameters are different than theirs). However, our point is not about this particular local optima, but that ES without exploration can get stuck indefinitely on some local optima and that novelty-driven exploration can help it get unstuck.

The Atari results illustrate that NS is an effective mechanism for encouraging directed exploration, given an appropriate behavior characterization, for complex, high-dimensional control tasks. A novelty pressure alone produces impressive performance on many games, sometimes even beating ES. Combining novelty and reward performs far better, and improves ES performance on tasks where it appears to get stuck on local optima. That being said, adding a novelty pressure is not always advantageous, especially when reward alone is sufficient.

GAME	ES	NS-ES	NSR-ES
ALIEN	4914.0	1600.0	2472.5
AMIDAR	462.0	168.7	255.8
BANK HEIST	230.0	110.0	213.0
BEAM RIDER [†]	815.2	900.0	823.6
BREAKOUT	13.5	10.8	124.3
FREEWAY [†]	31.8	22.7	33.4
FROSTBITE [†]	440.0	252.0	3326.0
GRAVITAR	1035.0	815.0	920.0
MONTEZUMA’S REVENGE	0.0	0.0	0.0
Q*BERT [†]	1425.0	10075.0	4160.0
SEAQUEST [†]	960.0	1615.0	2672.0
ZAXXON	11720.0	3810.0	7690.0

Table 1. ATARI 2600 reward of the highest performing policy found by each algorithm in any run. Scores are the mean over 10 stochastic policy evaluations, each of which has up to 30 random, initial no-operation actions. Table 2 shows results averaged across runs. Games with a [†] are those in which we observed ES to converge prematurely, presumably due to it encountering local optima.

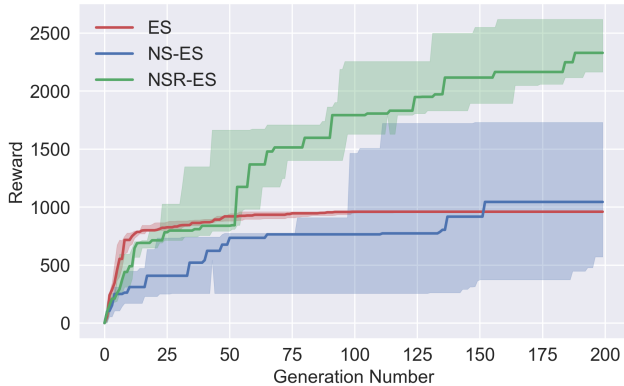


Figure 5. Avoidance of local optima in Seaquest. With our hyperparameter configuration, ES converges to a local optima it never escapes from, which involves not coming up for air. It never discovers that temporarily forsaking reward to surface for air ultimately can yield far larger rewards. By optimizing for both reward and novelty, NSR-ES is able to avoid local optima and achieve much higher performance than ES. With only exploratory pressure, NS-ES exhibits better performance in 3/5 runs, and would likely overtake ES in all runs with additional computation. Also note that, while its median score is similar to ES, it is not stuck on the same local optima, as it surfaces for air in all runs. NSR-ES is also still steadily improving at the end of training, suggesting that with longer training times, the gap in performance between NSR-ES and ES would grow.

GAME	ES	NS-ES	NSR-ES
ALIEN	3283.8	1124.5	2186.2
AMIDAR	462.0	134.7	255.8
BANK HEIST	140.0	50.0	130.0
BEAM RIDER [†]	871.7	805.5	876.9
BREAKOUT	5.6	9.8	10.6
FREEWAY [†]	31.1	22.8	32.3
FROSTBITE [†]	367.4	250.0	2978.6
GRAVITAR	1129.4	527.5	732.9
MONTEZUMA’S REVENGE	0.0	0.0	0.0
Q*BERT [†]	1075.0	1234.1	1400.0
SEAQUEST [†]	960.0	1044.5	2329.7
ZAXXON	9885.0	1761.9	6723.3

Table 2. ATARI 2600 median reward across runs. The scores are the median, across 5 runs, of the mean reward (over 30 stochastic evaluations) of each run’s final best policy. Plots of performance over time, along with bootstrapped confidence intervals of the median, for each algorithm for each game can be found in SI Sec. 6.8. Note that in some cases rewards reported here for ES are lower than those reported by Salimans et al. (2017), which could be due to differing hyperparameters (see SI Sec. 6.5). Games with a [†] are those in which we observed ES to converge prematurely, presumably due to it encountering local optima.

5. Discussion and Conclusion

NS and QD are a classes of evolutionary algorithms designed to avoid local optima and promote exploration in reinforcement learning environments, but have only been previously shown to work with small neural networks (on the order of hundreds of connections). ES was recently shown to be a viable evolutionary algorithm for training deep neural networks that can solve challenging, high-dimensional RL tasks (Salimans et al., 2017). It also is much faster when many parallel computers are available. Here we demonstrate that, when hybridized with ES, NS and QD not only preserve the attractive scalability properties of ES, but also help ES explore and avoid local optima in domains with deceptive reward functions. Our experimental results on the Humanoid Locomotion problems and Atari games illustrate that NS-ES alone can achieve high performance and avoid local optima. A QD algorithm we introduce that optimizes for novelty and reward, which we call NSR-ES, achieves even better performance, including superior performance to ES on a number of challenging domains. To the best of our knowledge, this paper reports the first attempt at augmenting ES to perform directed exploration in high-dimensional environments. We thus provide an option for those interested in taking advantage of the scalability of ES, but who also want higher performance on domains that have reward functions that are sparse or have local optima. The latter scenario will likely hold for most challenging, real-world domains that machine learning practitioners will wish to tackle in the future.

Additionally, this work highlights alternate options for exploration in RL domains. The first difference is to holistically describe the behavior of an agent instead of defining a per-state exploration bonus. The second is to encourage a population of agents to simultaneously explore different aspects of an environment. These styles of exploration are different than the state-based, single-agent exploration styles that are common in RL (Schmidhuber, 2010; Oudeyer & Kaplan, 2009), including recent extensions to deep RL (Pathak et al., 2017; Ostrovski et al., 2017; Tang et al., 2017). These new options thereby open new research areas into (1) comparing holistic vs. state-based exploration, and population-based vs. single-agent exploration, more systematically and on more domains, (2) investigating the best way to combine the merits of all of these options, and (3) hybridizing holistic and/or population-based exploration with other algorithms that work well on deep RL problems, such as policy gradients and DQN. It should be relatively straightforward to combine novelty search with policy gradients (NS-PG). It is less obvious how best to combine it with Q-learning to create NS-Q, but it may be possible and potentially fruitful.

As with any exploration method, encouraging novelty can

come at a cost if such an exploration pressure is not necessary. In Atari games such as Alien and Gravitar, and in the Humanoid Locomotion problem without a deceptive trap, both NS-ES and NSR-ES perform worse than ES. These results motivate research into how best to encourage novelty (and exploration more generally) only when needed. More generally, much work remains in terms of inventing more sophisticated methods to combine pressures for exploration and reward maximization within population-based exploration, which we consider an exciting area for future work.

Acknowledgements

We thank all of the members of Uber AI Labs, in particular Thomas Miconi, Rui Wang, Peter Dayan, and Theofanis Karaletsos, for helpful discussions. We also thank Justin Pinkul, Mike Deats, Cody Yancey, Joel Snow, Leon Rosenshein and the entire OpusStack Team inside Uber for providing our computing platform and for technical support.

References

- Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. In *NIPS*, pp. 1471–1479, 2016.
- Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 47:253–279, 2013.
- Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym, 2016.
- Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. Robots that can adapt like animals. *Nature*, 521:503–507, 2015. doi: 10.1038/nature14422.
- Cully, Antoine and Mouret, Jean-Baptiste. Behavioral repertoire learning in robotics. In *GECCO*, pp. 175–182, 2013.
- Dauphin, Yann, Pascanu, Razvan, Gülçehre, Çağlar, Cho, Kyunghyun, Ganguli, Surya, and Bengio, Yoshua. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *ArXiv e-prints*, abs/1406.2572, 2014.
- French, Robert M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.
- Houthoofd, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Vime: Variational information maximizing exploration. In *NIPS*, pp. 1109–1117, 2016.
- Huizinga, Joost, Mouret, Jean-Baptiste, and Clune, Jeff. Does aligning phenotypic and genotypic modularity improve the evolution of neural networks? In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference (GECCO)*, pp. 125–132, 2016.
- Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pp. 448–456, 2015.
- Jaderberg, Max, Dalibard, Valentin, Osindero, Simon, Czarnecki, Wojciech M, Donahue, Jeff, Razavi, Ali, Vinyals, Oriol, Green, Tim, Dunning, Iain, Simonyan, Karen, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.
- Kawaguchi, Kenji. Deep learning without poor local minima. In *NIPS*, pp. 586–594, 2016.
- Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- Kirkpatrick, James, Pascanu, Razvan, Rabinowitz, Neil, Veness, Joel, Desjardins, Guillaume, Rusu, Andrei A, Milan, Kieran, Quan, John, Ramalho, Tiago, Grabska-Barwinska, Agnieszka, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, pp. 201611835, 2017.
- Lange, Sascha and Riedmiller, Martin. Deep auto-encoder neural networks in reinforcement learning. In *IJCNN*, pp. 1–8, 2010.
- Lehman, Joel and Stanley, Kenneth O. Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX (GPTP 2011)*, 2011a.
- Lehman, Joel and Stanley, Kenneth O. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011b.
- Lehman, Joel and Stanley, Kenneth O. Evolving a diversity of virtual creatures through novelty search and local competition. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218, 2011c.
- Liepins, Gunar E. and Vose, Michael D. Deceptiveness and genetic algorithm dynamics. Technical Report CONF-9007175-1, Oak Ridge National Lab., TN (USA); Tennessee Univ., Knoxville, TN (USA), 1990.

- Miikkulainen, Risto, Liang, Jason, Meyerson, Elliot, Rawal, Aditya, Fink, Dan, Francon, Olivier, Raju, Bala, Navruzyan, Arshak, Duffy, Nigel, and Hodjat, Babak. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.
- Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *ICML*, pp. 1928–1937, 2016.
- Mouret, Jean-Baptiste and Clune, Jeff. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.
- Naddaf, Yavar. Game-independent ai agents for playing atari 2600 console games. 2010.
- Ostrovski, Georg, Bellemare, Marc G, Oord, Aaron van den, and Munos, Rémi. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.
- Oudeyer, Pierre-Yves and Kaplan, Frederic. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurobotics*, 1:6, 2009.
- Paquette, Phillip. Super mario bros. in openai gym, 2016.
- Pathak, Deepak, Agrawal, Pulkit, Efros, Alexei A, and Darrell, Trevor. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*, 2017.
- Petroski Such, Felipe, Madhavan, Vashisht, Conti, Edoardo, Lehman, Joel, Stanley, Kenneth O., and Clune, Jeff. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint to appear*, 2017.
- Pugh, Justin K, Soros, Lisa B, Szerlip, Paul A, and Stanley, Kenneth O. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 967–974, 2015.
- Pugh, Justin K, Soros, Lisa B., and Stanley, Kenneth O. Quality diversity: A new frontier for evolutionary computation. 3(40), 2016. ISSN 2296-9144.
- Rechenberg, Ingo. Evolutionsstrategien. In *Simulationmethoden in der Medizin und Biologie*, pp. 83–114. 1978.
- Rusu, Andrei A, Colmenarejo, Sergio Gomez, Gulcehre, Caglar, Desjardins, Guillaume, Kirkpatrick, James, Pascanu, Razvan, Mnih, Volodymyr, Kavukcuoglu, Koray, and Hadsell, Raia. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.
- Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. In *NIPS*, pp. 2234–2242, 2016.
- Salimans, Tim, Ho, Jonathan, Chen, Xi, and Sutskever, Ilya. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- Schmidhuber, Jürgen. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.
- Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael, and Moritz, Philipp. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015.
- Sehnke, Frank, Osendorfer, Christian, Rückstieß, Thomas, Graves, Alex, Peters, Jan, and Schmidhuber, Jürgen. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.
- Stadie, Bradley C, Levine, Sergey, and Abbeel, Pieter. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.
- Stanton, Christopher and Clune, Jeff. Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. *PloS one*, 2016.
- Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. 1998.
- Tang, Haoran, Abbeel, Pieter, Foote, Davis, Duan, Yan, Chen, OpenAI Xi, Houthoofd, Rein, Stooke, Adam, and DeTurck, Filip. # exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, pp. 2750–2759, 2017.
- van den Oord, Aaron, Kalchbrenner, Nal, Espeholt, Lasse, Vinyals, Oriol, Graves, Alex, et al. Conditional image generation with pixelcnn decoders. In *NIPS*, pp. 4790–4798, 2016.

Velez, Roby and Clune, Jeff. Novelty search creates robots with general skills for exploration. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pp. 737–744, 2014.

Velez, Roby and Clune, Jeff. Diffusion-based neuro-modulation can eliminate catastrophic forgetting in simple neural networks. *arXiv preprint arXiv:1705.07241*, 2017.

Wierstra, Daan, Schaul, Tom, Peters, Jan, and Schmidhuber, Juergen. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 3381–3387, 2008.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

6. Supplementary Information

6.1. Videos of agent behavior

Videos of example agent behavior in all the environments can be viewed here: <https://goo.gl/cVUG2U>.

6.2. Cross-sectional maze

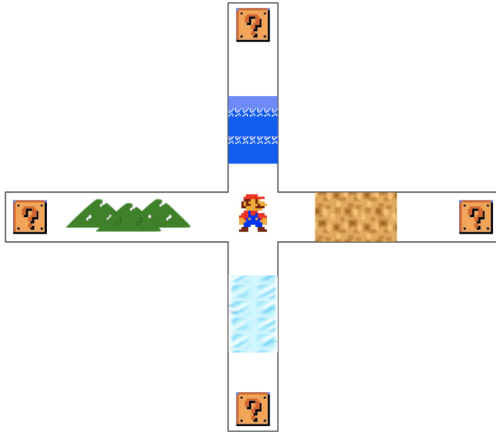


Figure 6. Hypothetical Hard Exploration Maze. In this maze, the agent needs to traverse 4 different terrains to obtain rewards associated with the “?” boxes. Traversing each terrain requires learning a certain skill (i.e. climbing, swimming, etc.). The sprites are from a Super Mario Bros. environment introduced by (Paquette, 2016).

6.3. NS-ES and NSR-ES Algorithm

Algorithm 1 NS-ES

```

1: Input: learning rate  $\alpha$ , noise standard deviation  $\sigma$ ,
   number of policies to maintain  $M$ , iterations  $T$ , be-
   havior characterization  $b(\pi_\theta)$ 
2: Initialize:  $M$  randomly initialized policy parameter
   vectors  $\{\theta_0^1, \theta_0^2, \dots, \theta_0^M\}$ , archive  $A$ , number of work-
   ers  $n$ 
3: for  $j = 1$  to  $M$  do
4:   Compute  $b(\pi_{\theta_0^j})$ 
5:   Add  $b(\pi_{\theta_0^j})$  to  $A$ 
6: end for
7: for  $t = 0$  to  $T - 1$  do
8:   Sample  $\theta_t^m$  from  $\{\theta_t^1, \theta_t^2, \dots, \theta_t^M\}$  via eq. 1
9:   for  $i = 1$  to  $n$  do
10:    Sample  $\epsilon_i \sim \mathcal{N}(0, \sigma^2 I)$ 
11:    Compute  $\theta_t^{i,m} = \theta_t^m + \epsilon_i$ 
12:    Compute  $b(\pi_{\theta_t^{i,m}})$ 
13:    Compute  $N_i = N(\theta_t^{i,m}, A)$ 
14:    Send  $N_i$  from each worker to coordinator
15:   end for
16:   Set  $\theta_{t+1}^m = \theta_t^m + \alpha \frac{1}{W\sigma} \sum_{i=1}^W N_i \epsilon_i$ 
17:   Compute  $b(\pi_{\theta_{t+1}^m})$ 
18:   Add  $b(\pi_{\theta_{t+1}^m})$  to  $A$ 
19: end for

```

Algorithm 2 NSR-ES

```

1: Input: learning rate  $\alpha$ , noise standard deviation  $\sigma$ ,
   number of policies to maintain  $M$ , iterations  $T$ , be-
   havior characterization  $b(\pi_\theta)$ 
2: Initialize:  $M$  sets of randomly initialized policy pa-
   rameters  $\{\theta_0^1, \theta_0^2, \dots, \theta_0^M\}$ , archive  $A$ , number of work-
   ers  $n$ 
3: for  $j = 1$  to  $M$  do
4:   Compute  $b(\pi_{\theta_0^j})$ 
5:   Add  $b(\pi_{\theta_0^j})$  to  $A$ 
6: end for
7: for  $t = 0$  to  $T - 1$  do
8:   Sample  $\theta_t^m$  from  $\{\theta_t^0, \theta_t^1, \dots, \theta_t^M\}$  via eq. 1
9:   for  $i = 1$  to  $n$  do
10:    Sample  $\epsilon_i \sim \mathcal{N}(0, \sigma^2 I)$ 
11:    Compute  $\theta_t^{i,m} = \theta_t^m + \epsilon_i$ 
12:    Compute  $b(\pi_{\theta_t^{i,m}})$ 
13:    Compute  $N_i = N(\theta_t^{i,m}, A)$ 
14:    Compute  $F_i = f(\theta_t^{i,m})$ 
15:    Send  $N_i$  and  $F_i$  from each worker to coordinator
16:   end for
17:   Set  $\theta_{t+1}^m = \theta_t^m + \alpha \frac{1}{W\sigma} \sum_{i=1}^W \frac{N_i + F_i}{2} \epsilon_i$ 
18:   Compute  $b(\pi_{\theta_{t+1}^m})$ 
19:   Add  $b(\pi_{\theta_{t+1}^m})$  to  $A$ 
20: end for

```

6.4. Preserving scalability

As shown in Salimans et al. (2017), ES scales well with the amount of computation available. Specifically, as more CPUs are used, training times reduce almost linearly, whereas DQN and A3C are not amenable to massive parallelization. NS-ES and NSR-ES, however, enjoy the same parallelization benefits as ES because they use an almost identical optimization process. The addition of an archive between agents in the meta-population does not hurt scalability because A is only updated after θ_t^m has been updated. Since A is kept fixed during the calculation of $N(\theta_t^{i,m}, A)$ and $f(\theta_t^{i,m})$ for all $i = 1 \dots n$ perturbations, the coordinator only needs to broadcast A once at the beginning of each generation. In all algorithms, the parameter vector θ_t^i must be broadcast at the beginning of each generation and since A generally takes up much less memory than the parameter vector, broadcasting both would incur effectively zero extra network overhead. NS-ES and NSR-ES do however introduce an additional computation conducted on the coordinator node. At the start of every generation we must compute the novelty of each candidate $\theta_t^m; m \in \{1, \dots, M\}$. For an archive of length n this operation is $O(Mn)$, but since M is small and fixed throughout training this cost is not significant in practice. Additionally, there are methods for keeping the archive small if this computation becomes an issue (Lehman & Stanley, 2011b).

6.5. Atari training details

Following Salimans et al. (2017), the network architecture for the Atari experiments consists of 2 convolutional layers (16 filters of size 8x8 with stride 4 and 32 filters of size 4x4 with stride 2) followed by 1 fully-connected layer with 256 hidden units, followed by a linear output layer with one neuron per action. The action space dimensionality can range from 3 to 18 for different games. ReLU activations are placed between all layers, right after virtual batch normalization units (Salimans et al., 2016). Virtual batch normalization is equivalent to batch normalization (Ioffe & Szegedy, 2015), except that the layer normalization statistics are computed from a reference batch chosen at the start of training. In our experiments, we collected a reference batch of size 128 at the start of training, generated by random agent gameplay. Without virtual batch normalization, Gaussian perturbations to the network parameters tend to lead to single-action policies. The lack of action diversity in perturbed policies cripples learning and leads to poor results (Salimans et al., 2017).

The preprocessing is identical to that in Mnih et al. (2016). Each frame is downsampled to 84x84 pixels, after which it is converted to grayscale. The actual observation to the network is a concatenation of 4 subsequent frames. There is a *frameskip* of 4. Each episode of training starts with up

to 30 random, no-operation actions.

For all experiments, we fixed the training hyperparameters for fair comparison. Each network is trained with the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $\eta = 10^{-2}$ and a noise standard deviation of $\sigma = 0.02$. The number of samples drawn from the population distribution each generation was $W = 5000$. For NS-ES and NSR-ES, we set $M = 3$ as the meta-population size and $k = 10$ for the nearest-neighbor computation, values that were both chosen through an informal hyperparameter search. We trained ES, NS-ES, and NSR-ES for a the same number of generations T for each game. The value of T varies between 150 and 300 depending on the number of timesteps per episode of gameplay (i.e. games with longer episodes are trained for 150 generations and vice versa). The figures in SI Sec. 6.8 show how many generations of training occurred for each game.

6.6. Humanoid Locomotion problem training details

The network architecture for the Humanoid Locomotion experiments is a multilayer perceptron with two hidden layers containing 256 neurons (with *tanh* activations) resulting in a network with 166,700 parameters. This architecture is the one in the configuration file included in the source code released by Salimans et al. (2017). The architecture described in their paper is similar, but smaller, having 64 neurons per layer Salimans et al. (2017).

For all experiments, we fixed the training hyperparameters for fair comparison. Each network was trained with the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $\eta = 10^{-2}$ and a noise standard deviation of $\sigma = 0.02$. The number of samples drawn from the population distribution each generation was $W = 10000$. For NS-ES and NSR-ES, we set $M = 5$ as the meta-population size and $k = 10$ for the nearest-neighbor computation, values that were both chosen through an informal hyperparameter search. We trained ES, NS-ES, and NSR-ES for a the same number of generations T for each game. The value of T is 600 for the Humanoid Locomotion problem and 800 for the Humanoid Locomotion with Deceptive Trap problem.

6.7. Humanoid Locomotion problem tabular results

ENVIRONMENT	ES	NS-ES	NSR-ES
ISOTROPIC	7767.8	5355.5	6891.5
DECEPTIVE	5.3	14.1	29.4

Table 3. Final results for the Humanoid Locomotion problem. The reported scores are computed by taking the median over 10 independent runs of the rewards of the highest scoring policy per run (each of which is the mean over ~ 30 evaluations).

6.8. Plots of Atari learning across training (generations)

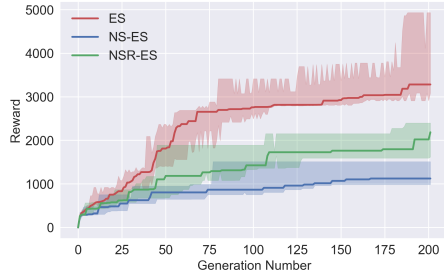


Figure 7. Results for the game Alien.

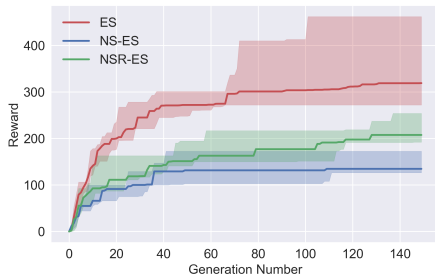


Figure 8. Results for the game Amidar.

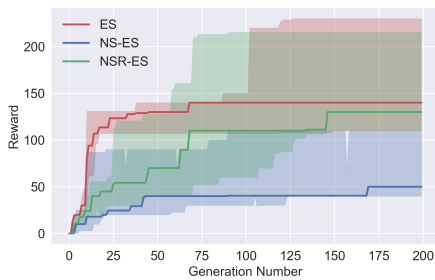


Figure 9. Results for the game Bank Heist.

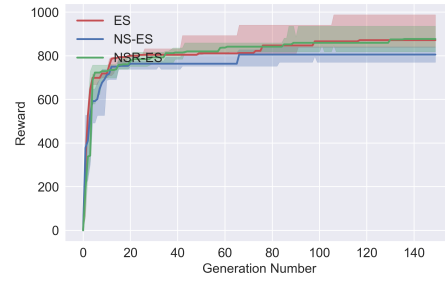


Figure 10. Results for the game Beam Rider.

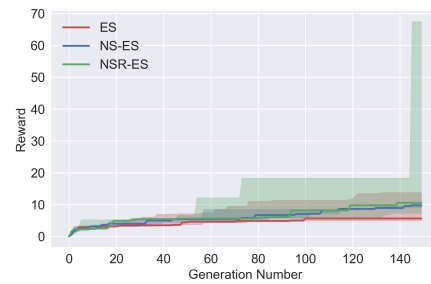


Figure 11. Results for the game Breakout.

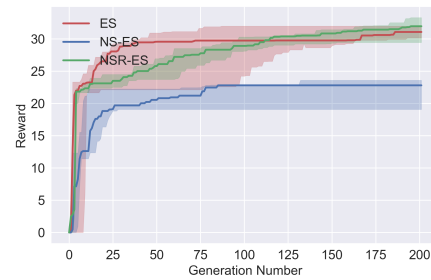


Figure 12. Results for the game Freeway.

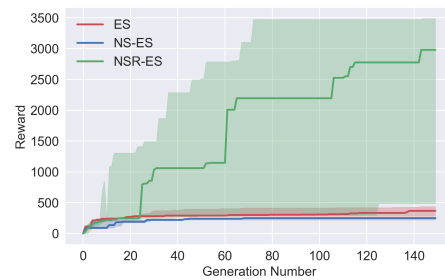


Figure 13. Results for the game Frostbite.

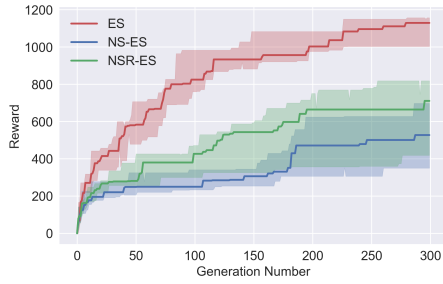


Figure 14. Results for the game Gravitar.

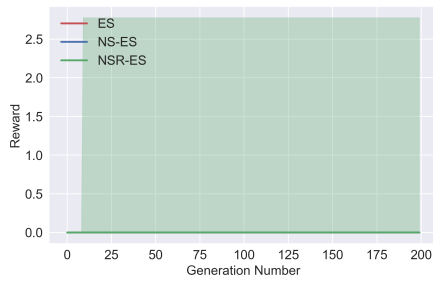


Figure 15. Results for the game Montezuma's Revenge.

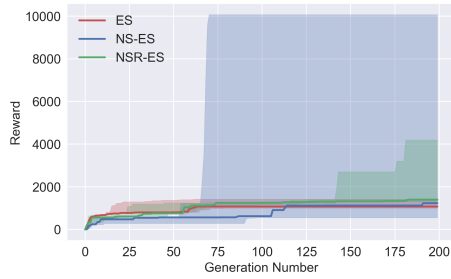


Figure 16. Results for the game Q*Bert.

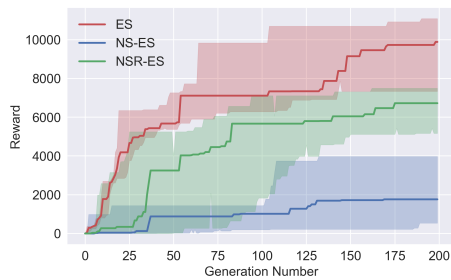


Figure 17. Results for the game Zaxxon.

6.9. Overhead plots of agent behavior on the Humanoid Locomotion with Deceptive Trap Problem.

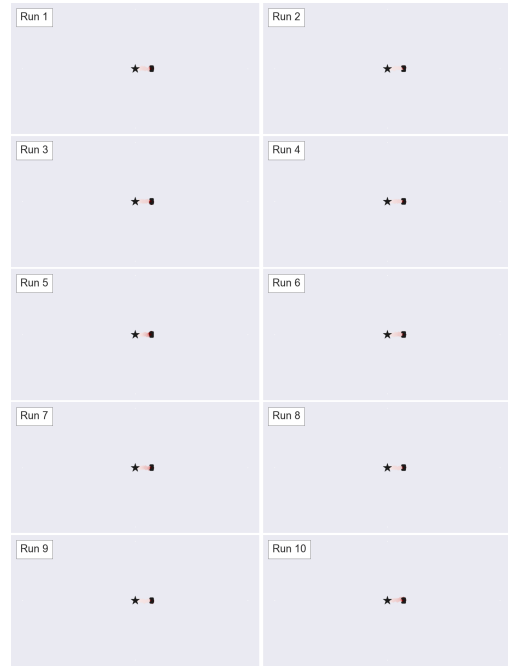


Figure 18. Overhead plot of ES across 10 independent runs on the Humanoid Locomotion with Deceptive Trap problem.

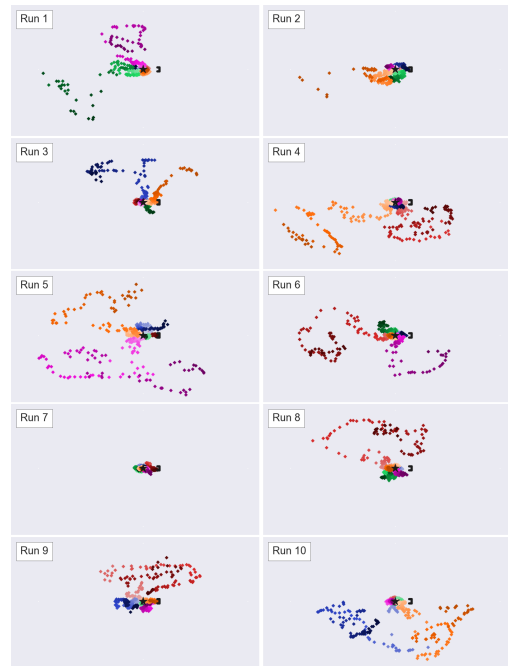


Figure 19. Overhead plot of NS-ES across 10 independent runs on the Humanoid Locomotion with Deceptive Trap problem.

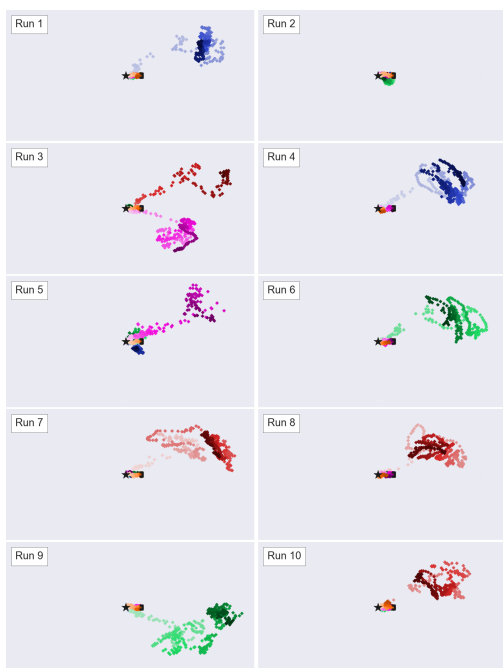


Figure 20. Overhead plot of NSR-ES across 10 independent runs on the Humanoid Locomotion with Deceptive Trap problem.