Evolvability ES: Scalable Evolutionary Meta-Learning

Alexander Gajewski[1*] **Jeff Clune**[2] **Kenneth O. Stanley**[2] **Joel Lehman**[2]
[1] Columbia University [2] Uber AI Labs

Abstract

This paper introduces *Evolvability ES*, an evolutionary meta-learning algorithm for deep reinforcement learning that scales well with computation and to models with many parameters (e.g. deep neural networks). This algorithm produces parameter vectors for which *random perturbations* yield a repertoire of diverse behaviors; we highlight that surprisingly, such points in the search space exist and can be reliably found. The resultant algorithm is evocative of MAML (which optimizes parameters to be amenable to quick further gradient descent), but operates in a much different way, and provides a different profile of benefits and costs. We highlight Evolvability ES's potential through experiments in 2-D and 3-D locomotion tasks, where surprisingly evolvable solutions are found that can be quickly adapted to solve an unseen test-task. This work thus opens up a novel research direction in exploiting the potential of evolvable representations for deep neural networks.

1 Introduction

Biology provides multiple perspectives on meta-learning, i.e. learning to learn. From one perspective, biological evolution can be viewed as the meta-learning process that crafted the ability of humans to flexibly learn across our individual lifetimes. In other words, evolution (i.e. the meta-learning algorithm) crafted our plastic brains (i.e. our learning algorithm). Algorithms that train recurrent neural networks (RNNs) to meta-learn RL algorithms [16, 3] or neural network (NN) learning rules [11] can be seen in this light, as algorithms that create NNs that can adapt to task-specific challenges.

This paper focuses on a different analogy: evolution also rewards the ability to quickly adapt *over generations of evolution*. For example, a prey species may benefit from quickly adjusting to an adaptation of its predator, and vice versa. In biology, this ability for an organism to further evolve is called its *evolvability* [8, 15], a subject of much study also within evolutionary computation [10, 1], where its inspiration is intriguingly similar to that of the meta-learning community. One framing of evolvability is to search for setting(s) of parameters from which perturbations will generate a usefully diverse set of solutions, allowing an individual or population to quickly adapt to new challenges [10, 18]. The connection between this thread of research and meta-learning in general offers a fruitful opportunity for cross-pollination so far not exploited.

A previous algorithm to encourage this form of evolvability, called Evolvability Search [10], directly searches for evolvable parameter vectors, but requires exhaustive computation. Taking inspiration from Evolvability Search and a recent evolutionary strategy (ES) algorithm [13], we extend stochastic computation graphs [14] to enable an ES-based algorithm that optimizes evolvability, which we call *Evolvability ES*. This approach allows searching for evolvability at deep network scales, in a way that easily benefits from additional computation. We show the potential of Evolvability ES through a series of experiments in 2-D and 3-D simulated robot locomotion tasks, demonstrating that perturbations of an evolved parameter vector can yield a surprising diversity of outcomes, and can be quickly adapted to solve a particular task. By exposing the existence of such points in the search space and introducing a principled approach to finding them, this work opens up a novel research direction in exploiting the potential of evolvable representations in the era of deep learning.

[*] Work done while at Uber AI Labs

2nd Workshop on Meta-Learning at NeurIPS 2018, Montréal, Canada.

2 Evolution Strategies

Natural Evolution Strategies (NES; [17]) is a black-box optimization algorithm designed for non-differentiable functions. Because the gradients of such a function $f(z)$ are unavailable, NES instead creates a smoother version of f that *is* differentiable, by defining a distribution $\pi(z;\theta)$ over its argument and setting the smoothed loss function $J(\theta) = \mathbb{E}_z\left[f(z)\right]$. This function is then optimized iteratively with gradient descent, where the gradients are estimated by samples from π.

Salimans et al. [13] showed recently that NES with an isotropic Gaussian distribution of fixed variance (i.e. $\pi = \mathcal{N}(\mu, \sigma I)$ and $\theta = \mu$) is competitive with deep reinforcement learning on high-dimensional reinforcement learning (RL) tasks, and can be very time-efficient because the expensive gradient estimation is easily parallelizable. Salimans et al. [13] refer to this algorithm as Evolution Strategies (ES), and we adopt their terminology in this work, referring to it as Standard ES. Making a biological analogy, π is can be viewed as the analogue of a *population* of individuals that evolves by one *generation* at every gradient step, where μ can be viewed as the parent individual, and samples from π can be termed that parent's *pseudo-offspring* cloud.

3 Stochastic Computation Graphs for ES-like Algorithms

In supplemental sections S3 and S4, we generalize stochastic computation graphs [14] to more easily account for nested expectations, and show that this new formalism yields *surrogate loss functions* that can automatically be differentiated by popular tools like PyTorch [12]. Through this approach, nearly any loss function involving potentially nested expectations over differentiable probability distributions can be automatically optimized with gradient descent through sampling. This approach is applied in the next section to enable an efficient evolvability-optimizing variant of ES.

4 Evolvability ES

The approach described here searches for parameter vectors whose perturbations lead to diverse behaviors, and which therefore may be quickly adapted to new tasks after training. This adaptability-based training relates conceptually to Model-Agnostic Meta-Learning (MAML; [5]), wherein a single set of parameters is trained such that a few additional SGD training steps will adapt it to a new task. However, MAML requires (1) differentiating through an optimization procedure, which is computationally expensive and prone to local optima, and (2) an explicit mechanism for sampling new tasks from a distribution, which may impose an inefficient curriculum for learning. In contrast, Evolvability ES (1) optimizes for adaptability by maximizing the effect of *random* perturbations, thereby avoiding differentiating through optimization, and (2) instantiates a self-adapting learning curriculum: a solution is incentivized for its perturbations to increasingly span the space of behaviors in a bottom-up fashion (see also Eysenbach et al. [4]). As with ES [13], Evolvability ES scales easily and efficiently to make use of large-scale computation. We now formally describe the method.

Consider the isotropic Gaussian distribution π of ES. As in Evolvability Search [10], we wish to maximize some statistic of behavioral diversity of a parameter vector's perturbations. Formally, behavior is represented as a behavior characteristic (BC; [9]), a vector-valued function mapping parameter vectors z to behaviors $\mathbf{B}(z)$. For example, in a locomotion task, a policy's behavior might be represented as its final position on a plane. Here, we consider two diversity statistics which lead to maximum variance (MaxVar) and maximum entropy (MaxEnt) variants of Evolvability ES.

MaxVar Evolvability ES maximizes the total variation of the BC over the population, which can be formulated as the following loss function:

$$J(\theta) = \sum_j \mathbb{E}_z\left[(B_j(z) - \mu_j)^2\right], \tag{1}$$

where the expectation is over policies $z \sim \pi(\cdot;\theta)$, the summation is over components j of the BC, and μ_j represents the mean of the jth component of the BC.

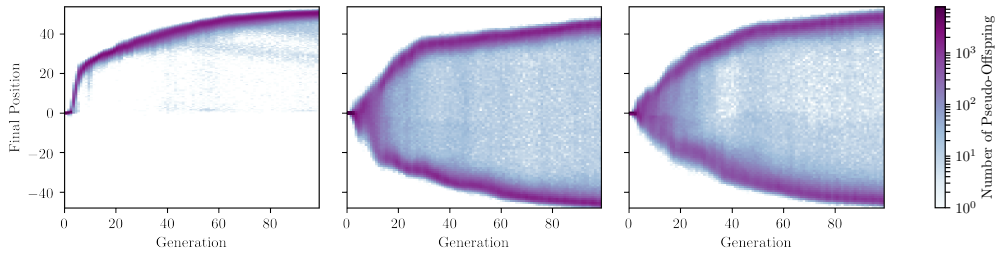
MaxEnt Evolvability ES maximizes entropy rather than variance. To estimate entropy, we first compute a kernel-density estimate of the distribution of behavior for some kernel function φ, giving

$$p(\boldsymbol{B}(z);\theta) \approx \mathbb{E}_{z'}[\varphi(\boldsymbol{B}(z') - z)], \tag{2}$$

which can be applied to derive a loss function which estimates the entropy:

$$J(\theta) = -\mathbb{E}_z\left[\log \mathbb{E}_{z'}[\varphi(\boldsymbol{B}(z') - z)]\right]. \tag{3}$$

In practice, these loss functions are differentiated with PyTorch (as described in supplemental section S4) and both the loss and their gradients are then estimated from samples.



(a) Standard ES (b) MaxVar Evolvability ES (c) MaxEnt Evolvability ES

Figure 1: **Distribution of behaviors over evolution in the 2-D locomotion domain.** Heat-maps of the final x positions of $10,000$ policies sampled from the population distribution are shown for each generation over training time. These plots suggest that both Evolvability ES methods discover policies that travel nearly as far both backwards and forwards as Standard ES travels forward alone.

5 Experiments

5.1 2-D Locomotion

In the 2-D locomotion task, a NN policy controls the Half-Cheetah robot from the PyBullet simulator [2]. The policy receives as input the robot's position and velocity, and the angle and angular velocity of its joints; the policy outputs as actions the torques to apply to each of the robot's joints. In this domain, we characterize behavior as the final x coordinate of the robot after a fixed number of steps of the simulation. Hyperparameters and algorithmic details can be found in supplemental section S2; see supplemental section S1 for experiments in a diagnostic setting.

Figure 1 shows the distribution of behavior over training time. First, these results show that, surprisingly, there exist policies such that small perturbations in parameter space result in diametrically opposed behaviors. Both variants perform roughly equivalently, despite the intuition that the MaxEnt version should demonstrate a distribution of behaviors that is spread more uniformly.

Supplemental Figure S2 a shows performance curves for all methods in this domain (for maximum distance travelled, the metric that Standard ES optimizes but that Evolvability ES optimizes for only indirectly). As expected (because it directly optimizes for this metric), Standard ES finds policies that locomote slightly further than Evolvability ES, though this is not true in every pair of runs.

5.2 3-D Locomotion

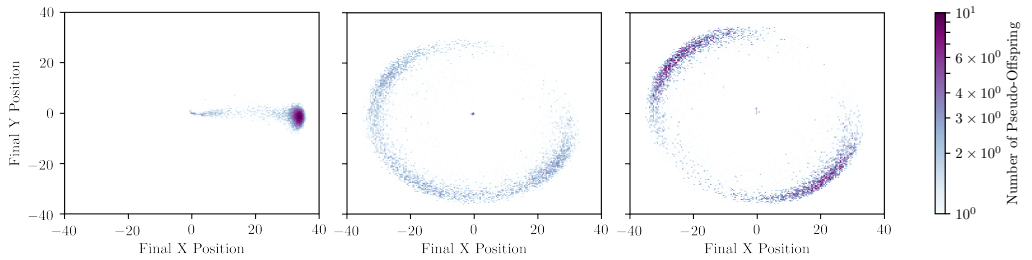
In the 3-D locomotion task, a policy controls the Ant robot from the PyBullet simulator [2], receiving as inputs the robot's position and velocity, and the angle and angular velocity of its joints; and outputting torques for each of its joints. In this domain, a policy's BC is its final x, y position.

Figure 2 shows the distributions of behavior after training. Interestingly, both variants form a ring of policies, i.e. the method finds a parameter vector from which nearly *all directions* of travel are reachable through parameter perturbations. Note that for Evolvability ES there is low density of behaviors in the interior of the ring (i.e. few mutations are degenerate), whereas Standard ES exhibits a moderate-density trail of reduced performance (i.e. its policy is less robust to mutation).

Figure S2 shows the final x position of the policy that moved farthest in the x direction, among $10,000$ individuals sampled from the population each generation over training time. Interestingly, in this domain, Evolvability ES sometimes finds policies which *move farther* than those found by Standard ES, though it is has no direct pressure to move farther (and Standard ES does).

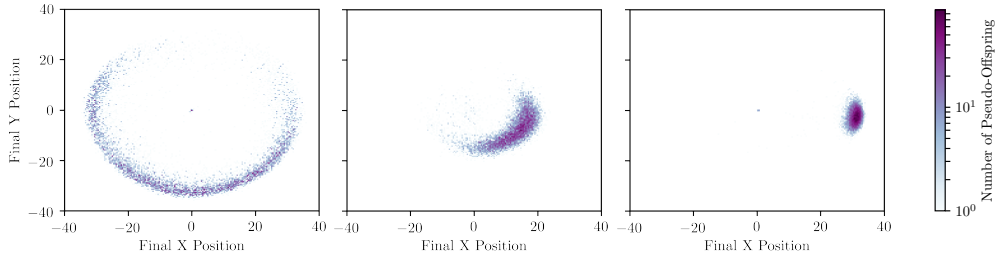
5.3 Fast Adaptation in the 3-D Locomotion Domain

We next test adaptation to new tasks, similar to previous experiments with MAML [5]. As before, we first train Evolvability ES in the the 3-D locomotion domain. Using such trained populations as initializations for Standard ES, we then set as an explicit objective for the agent to walk in a particular direction, here, for simplicity, to travel as far as possible along the positive x axis. Note that during initial training the algorithm had no information about this specific goal.



(a) Standard ES (b) MaxVar Evolvability ES (c) MaxEnt Evolvability ES

Figure 2: **Distribution of behaviors after training in the 3-D locomotion domain.** Heat-maps of the final positions of $10,000$ policies sampled from the population distribution at generation 100. These plots suggest that both Evolvability ES variants successfully find policies which move in many different directions, and roughly as far as Standard ES travels in the positive x direction alone.



(a) Before Adaptation (b) After 5 Generations (c) After 20 Generations

Figure 3: **Distribution of behaviors during adaptation in the 3-D locomotion domain.** Heat-maps are shown of the final positions of $10,000$ policies sampled from the population distribution initialized with MaxEnt Evolvability ES, and adapted to move in the positive x direction with Standard ES over several generations. These plots suggest that MaxEnt Evolvability ES successfully finds policies that can quickly adapt to perform new tasks. See Figure S3 for the MaxVar version.

Figures 3 and S3 show heat-maps of behavior characteristics for the pre-trained populations produced by Evolvability ES, as well as following a small number of Standard ES updates. The population successfully converges in accordance with the new selection pressure, and importantly, it adapts much more quickly than does a randomly initialized population, as shown in Figure S4. The conclusion is that Evolvability ES is a viable meta-learning algorithm, applicable to similar situations as is MAML.

5.4 Meta-Learning Specialists

While some applications require only a single meta-learned policy, other applications may require meta-learning *multiple* policies, each *specialized* to adapt to particular families of tasks [18]. To study this, we compare two multi-modal variants of Evolvability ES, where the different modes of a Gaussian Mixture Model (GMM; [7]) distribution can learn to specialize. In the first variant (vanilla GMM), we run Evolvability ES with a multi-modal GMM population, where each mode is equally likely and is separately randomly initialized. In the second variant (splitting GMM), we first train a uni-modal population with Evolvability ES until convergence, then seed a GMM from it, by initializing the new component means from independent samples from the pre-trained uni-modal population to break symmetry. Figures S5 and S6 compare the qualitative performance of these two variants; in this domain, the splitting GMM more efficiently evolves complementary specialists.

6 Conclusion

We have presented Evolvability ES, a novel and scalable algorithm for evolving populations that span diverse behaviors and that may be quickly adapted to perform new tasks. Experiments in 2D and 3D locomotion domains demonstrate the algorithm's potential, and surprisingly reveal that points in the search space exist from which *random* parameter perturbations yield a wide diversity of behavior; the conclusion is that Evolvability ES is a new addition to the toolbox of meta-learning algorithms and opens up a new research direction for exploring evolvable representations for deep NNs.

References

[1] Lee Altenberg et al. The evolution of evolvability in genetic programming. *Advances in genetic programming*, 3:47–74, 1994.

[2] E Coumans and Y Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[3] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[4] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *CoRR*, abs/1802.06070, 2018. URL `http://arxiv.org/abs/1802.06070`.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[6] Michael C. Fu. Chapter 19 gradient estimation. *Simulation Handbooks in Operations Research and Management Science*, 13:575, 2006. doi: 10.1016/s0927-0507(06)13019-4.

[7] DP Geoffrey McLachlan. Finite mixture models, 2000.

[8] Marc Kirschner and John Gerhart. Evolvability. *Proceedings of the National Academy of Sciences*, 95(15):8420–8427, 1998.

[9] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[10] Henok Mengistu, Joel Lehman, and Jeff Clune. Evolvability search. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO 16*, 2016. doi: 10.1145/2908812.2908838.

[11] Thomas Miconi, Jeff Clune, and Kenneth O Stanley. Differentiable plasticity: training plastic neural networks with backpropagation. *arXiv preprint arXiv:1804.02464*, 2018.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[13] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

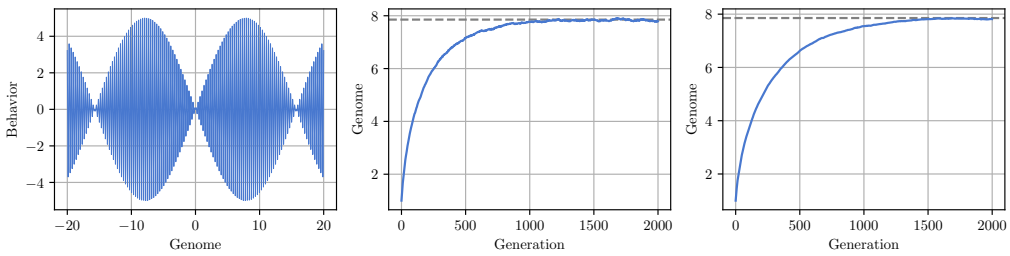
[14] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *CoRR*, abs/1506.05254, 2015. URL `http://arxiv.org/abs/1506.05254`.

[15] Günter P Wagner and Lee Altenberg. Perspective: complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996.

[16] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[17] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, and Jürgen Schmidhuber. Natural evolution strategies, 2011.

[18] Bryan Wilder and Kenneth Stanley. Reconciling explanations for the evolution of evolvability. *Adaptive Behavior*, 23(3):171–179, 2015.



(a) Interference Pattern (b) MaxVar Evolvability ES (c) MaxEnt Evolvability ES

Figure S1: **Interference Pattern Results.** In the interference pattern task, a solution consists of a single floating point parameter, which corresponds to the x coordinate on the interference pattern shown in (a). An individual's behavior characteristic is then mapped as the corresponding y coordinate on the interference pattern. The training plots shown in (b) and (c) validate both Evolvablity ES variants, showing that they converge to the point with behavior that is most sensitive to perturbations, shown above with a dashed line.

Supplemental Information

Included in the supplemental information is a simple explanatory task for better understanding Evolvability ES (section S1); experimental details and hyperparameters for all algorithms (section S2); a description of stochastic computation graphs and how we extended them (sections S3 and S4); and the particular stochastic computation graphs that enable calculating loss and gradients for ES and Evolvability ES (section S5).

S1 Interference Pattern Task

To validate the Evolvability ES approach, this illustrative experiment introduces a simple optimization problem. Consider the interference pattern in Figure S1a. In this figure, the x-axis represents the 1-dimensional parameter space, and the y-axis represents the 1-dimensional behavior characterization. Intuitively, an evolvability-seeking algorithm should find parameter settings such that small parameter perturbations will result in diverse behaviors. In this task, the most evolvable point in the search space is $\frac{\pi}{2*0.2} \approx 7.9$, where perturbations will have the largest effect on behavior.

Indeed, Figure S1 shows that both variants of Evolvability ES approach the optimal value in parameter space as training progresses. Interestingly, if in this domain the true goal were to maximize y (instead of maximizing its variance), Evolvability ES would outperform Standard ES, since the gradient of $\mathbb{E}[y]$ with respect to the mean of a distribution with standard deviation larger than the period of the faster sine wave will be approximately zero.

S2 Experimental Details

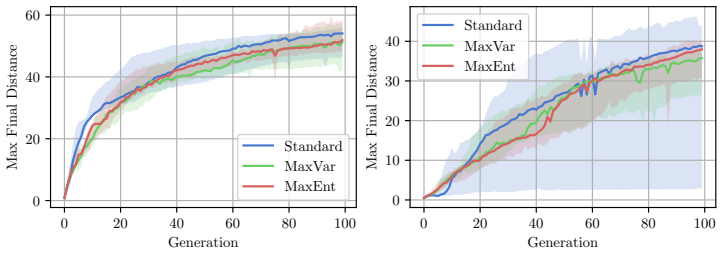
This section presents additional plots useful for better understanding the performance of Evolvability ES, as well as further details and hyperparameters for all algorithms.

S2.1 Additional Plots

Figure S2 shows training curves for algorithms in both of the locomotion domains.

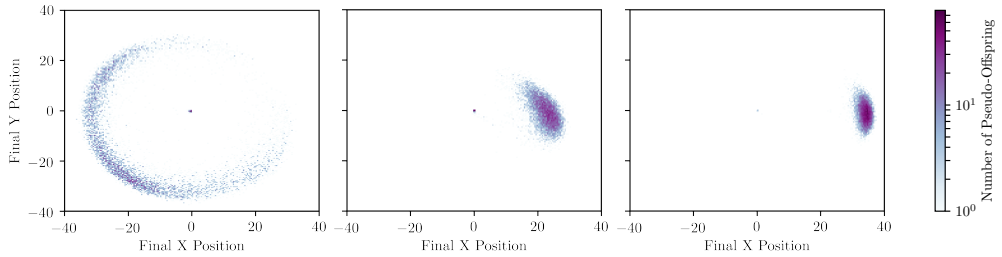
Figure S3 highlights how the distribution of behaviors changes during meta-learning test-time to quickly adapt to the task at hand for MaxVar Evolvability ES (see Figure 3 for the MaxEnt version), and Figure S4 shows the advantage of Evolvability ES for fast adaptation over training from scratch.

Figures S5 and S6 contrast the two variants of multi-modal Evolvability ES.



(a) 2D Locomotion (b) 3D Locomotion

Figure S2: **Evolvability Training Curves.** Comparison between Standard ES, MaxVar Evolvability ES, and MaxEnt Evolvability ES on (a) the 2D locomotion task, and (b) the 3D locomotion task. Both plots show the *maximum* final distance from the origin over $10,000$ samples from the population distribution during training. Median and range over 6 runs shown. These plots show that both variants of Evolvability ES typically find policies which move almost as far as Standard ES on both domains, despite moving in many more directions than Standard ES.



(a) Before Adaptation (b) After 5 Generations (c) After 20 Generations

Figure S3: **Distribution of behaviors during adaptation in the 3D locomotion domain.** Heatmaps of the final positions of $10,000$ policies sampled from the population distribution initialized with MaxVar Evolvability ES, and adapted to move in the positive x direction with Standard ES over several generations. These plots suggest that MaxVar Evolvability ES successfully finds policies which can quickly adapt to perform new tasks. See Figure 3 for the MaxEnt version.

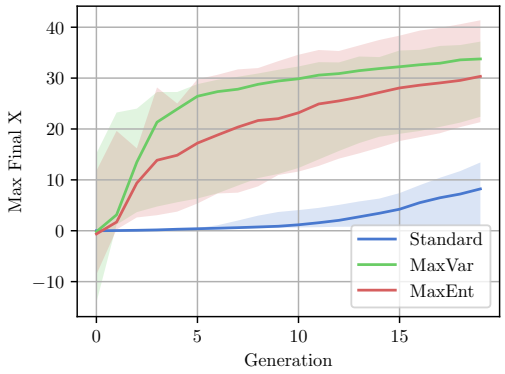
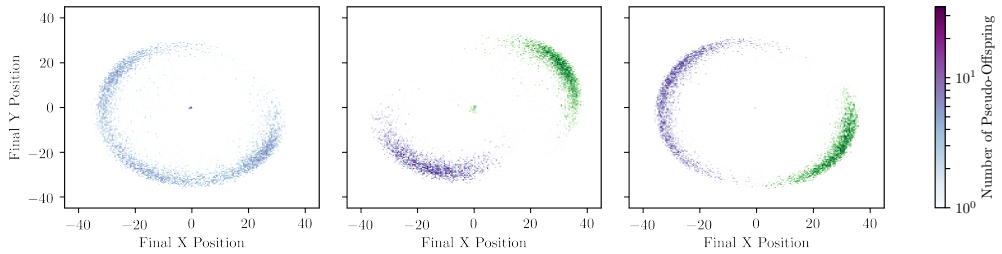
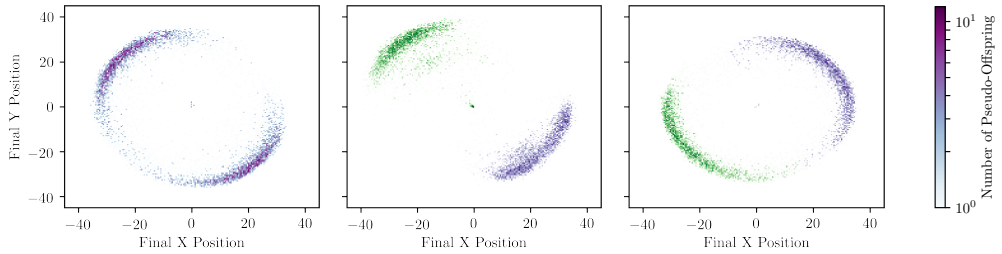


Figure S4: **Adaptation Training Curves.** Comparison between three different initial populations trained with Standard ES for only 20 generations, one initialized randomly, one trained for 100 generations with MaxVar Evolvability ES, and one trained for 100 generations with MaxEnt Evolvability ES. This plot shows the *maximum* x position over $10,000$ policies sampled from the population distribution during training. Median and range over 6 runs shown. This plot suggests that populations trained by both methods of Evolvability ES evolve much more quickly than randomly initialized population.



(a) Uni-Modal Evolvability ES (b) Vanilla GMM (c) Splitting GMM

Figure S5: **Distribution of behaviors for uni- and multi-modal MaxVar Evolvability ES variants**. Heat-maps of the final positions of $10,000$ policies sampled from the population distribution at generation 100 of MaxVar Evolvability ES, with (a) one component and (b) a vanilla GMM with two components. Also shown is the result of splitting the single component in (a) into two components and evolving for 20 additional generations. These plots suggest that the vanilla GMM fails to fill some of the behavior space, and that the splitting GMM is somewhat more successful.



(a) Uni-Modal Evolvability ES (b) Vanilla GMM (c) Splitting GMM

Figure S6: **Distribution of behaviors for uni- and multi-modal MaxEnt Evolvability ES variants**. Heat-maps are shown of the final positions of $10,000$ policies sampled from the population at generation 100 of MaxEnt Evolvability ES, with (a) one component and (b) a vanilla GMM with two components. Also shown is the result of (c) splitting the trained single component into two components, and evolving for 20 additional generations. These plots suggest that the splitting GMM is more successful at filling the behavior space.

S2.2 Hyperparameters and Training Details

For Standard ES, fitness was rank-normalized to take values symmetrically between -0.5 and 0.5 at each generation before computing gradient steps. For both variants of Evolvability ES, BCs were whitened to have mean zero and a standard deviation of one at each generation before computing losses and gradients. This was done instead of rank normalization in order to preserve density information for variance and entropy estimation.

A Gaussian kernel with standard deviation 1.0 was used for the MaxEnt variant of Evolvability ES to estimate the density of behavior given samples from the population distribution.

S2.2.1 Interference Pattern Details

The interference pattern was generated by the function

$$f(x) = 5 \sin \frac{x}{5} \sin 20x. \tag{4}$$

Hyperparameters for the interference pattern task are shown in Tables S1 and S2.

S2.2.2 Locomotion Task Details

For the locomotion tasks, all environments were run deterministically and no action noise was used during training. The only source of randomness was from sampling from the population distribution.

Hyperparameter	Setting
Learning Rate	0.05
Population Standard Deviation	0.5
Population Size	500

Table S1: **MaxVar Hyperparameters: Interference Pattern Task.**

Hyperparameter	Setting
Learning Rate	0.01
Population Standard Deviation	0.5
Population Size	500
Kernel Standard Deviation	1.0

Table S2: **MaxEnt Hyperparameters: Interference Pattern Task.**

Policies were executed in the environment for $1,000$ timesteps. For comparison to Evolvability ES, the fitness function for Standard ES was set to be the final x position of a policy, rewarding Standard ES for walking as far as possible in positive x direction.

NNs for both 2D and 3D locomotion were composed of two hidden layers with 256 hidden units each, resulting in 166.7K total parameters, and were regularized with L2 penalties during training. Inputs to the networks were normalized to have mean zero and a standard deviation of one based the mean and standard deviation of the states seen in a random subset of all training rollouts, with each rollout having probability 0.01 of being sampled.

Experiments were performed on a cluster system and were distributed across a pool of 550 CPU cores shared between two runs of the same algorithm. Each run took approximately 5 hours to complete.

Hyperparameters for both variants of Evolvability ES are shown Tables S3 and S4.

Hyperparameter	Setting
Learning Rate	0.01
Population Standard Deviation	0.02
Population Size	10,000
L2 Regularization Coefficient	0.05

Table S3: **MaxVar Hyperparameters: Locomotion Tasks.**

S3 Stochastic Computation Graphs

A stochastic computation graph, as defined in Schulman et al. [14], is a directed acyclic graph consisting of fixed input nodes, deterministic nodes representing functions of their inputs, and stochastic nodes representing random variables distributed conditionally on their inputs.

A stochastic computation graph $\mathcal{G}$ represents the expectation (over its stochastic nodes $\{z_i\}$) of the sum of its output nodes $\{f_i\}$, as a function of its input nodes $\{x_i\}$:

$$\mathcal{G}(x_1, \ldots, x_l) = \mathbb{E}_{z_1,\ldots,z_m}\left[\sum_{i=1}^{n} f_i\right] \tag{5}$$

For example, consider the stochastic computation graph in Figure S7, reproduced from [14]. This graph $\mathcal{G}$ represents the expectation

$$\mathcal{G}(x_0, \theta) = \mathbb{E}_{x_1,x_2}\left[f_1(x_1) + f_2(x_2)\right], \tag{6}$$

Hyperparameter	Setting
Learning Rate	0.01
Population Standard Deviation	0.02
Population Size	10,000
L2 Regularization Coefficient	0.05
Kernel Bandwidth	1.0

Table S4: **MaxEnt Hyperparameters: Locomotion Tasks.**

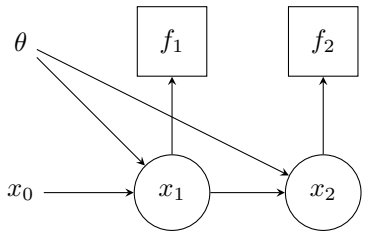


Figure S7: Example stochastic computation graph: input nodes are depicted with no border, deterministic nodes with square borders, and stochastic nodes with circular borders.

where $x_1 \sim p(\cdot; x_0, \theta)$ and $x_2 \sim p(\cdot; x_1, \theta)$.

A key property of stochastic computation graphs is that they may be differentiated with respect to their inputs. Using the score function estimator [6], we have that

$$\nabla_\theta \mathcal{G}(x_0, \theta) = \mathbb{E}_{x_1, x_2}\Big[\nabla_\theta \log p(x_1; \theta, x_0)(f_1(x_1) + f_2(x_2)) + \nabla_\theta \log p(x_2; \theta, x_1) f_2(x_2)\Big]. \quad (7)$$

Schulman et al. [14] also derive *surrogate* loss functions for stochastic computation graphs, allowing for implementations of stochastic computation graphs with existing automatic differentiation software.

For example, given a sample $\{x_1^i\}_{1 \le i \le N}$ of x_1 and $\{x_2^i\}_{1 \le i \le N}$ of x_2, we can write

$$\hat{L}(\theta) = \frac{1}{N} \sum_i \log p(x_1^i; \theta, x_0)(f_1(x_1^i) + f_2(x_2^i)) + \log p(x_2^i; \theta, x_1^i) f_2(x_2^i). \quad (8)$$

Now to estimate $\nabla_\theta \mathcal{G}(x_0, \theta)$, we have

$$\nabla_\theta \mathcal{G}(x_0, \theta) \approx \nabla_\theta \hat{L}(\theta), \quad (9)$$

which may be computed with popular automatic differentiation software.

S4 Nested Stochastic Computation Graphs

We make two changes to the stochastic computation graph formalism:

1. We add a third type of node which represents the expectation over one of its parent stochastic nodes of one of its inputs. We require that a stochastic node representing a random variable z be a dependency of exactly one expectation node over z, and that every expectation node over a random variable z depend on a stochastic node representing z.
2. Consider a stochastic node representing a random variable z conditionally dependent on a node y. Rather than expressing this as a dependency of z on y, we represent this as a dependency between the expectation node over z on y. Formally, this means all stochastic nodes are required to be leaves of the computation graph.

Because "nested stochastic computation graphs," as we term them, contain their expectations explicitly, they simply represent the sum of their output nodes (instead of the expected sum of their output nodes, as with regular stochastic computation graphs).

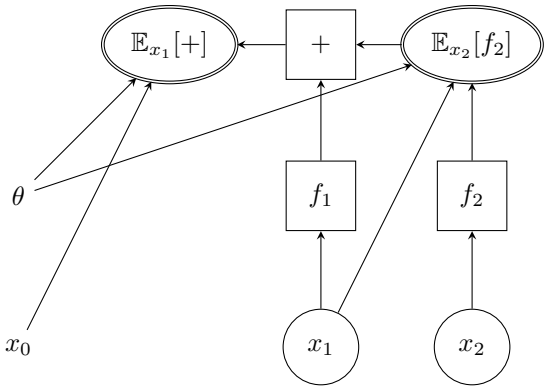


Figure S8: Example nested stochastic computation graph: input nodes are depicted with no border, deterministic nodes with square borders, stochastic nodes with circular borders, and expectation nodes with double elliptical borders.

As an example, consider the nested stochastic computation graph $\mathcal{G}$ depicted in Figure S8. First, note that $\mathcal{G}$ is indeed a nested stochastic computation graph, because the stochastic nodes and expectation nodes correspond, and because all stochastic nodes are leaves of the graph. Next, note that $\mathcal{G}$ is equivalent to the stochastic computation graph of Figure S7 in the sense that it computes the same function of its inputs:

$$\mathcal{G}(x_0, \theta) = \mathbb{E}_{x_1}\left[f_1(x_1) + \mathbb{E}_{x_2}\left[f_2(x_2)\right]\right] \tag{10}$$
$$= \mathbb{E}_{x_1, x_2}\left[f_1(x_1) + f_2(x_2)\right] \tag{11}$$

The original stochastic computation graph formalism has the advantage of more clearly depicting conditional relationships, but this new formalism has two advantages:

1. Nested stochastic computation graphs can represent arbitrarily nested expectations. We have already seen this in part with the example of Figure S8, but we shall see this more clearly in a few sections.
2. It is trivial to define surrogate loss functions for nested stochastic computation graphs. Moreover, these surrogate loss functions have the property that in the forward pass, they estimate the true loss function.

Consider a nested stochastic computation graph $\mathcal{G}$ with input nodes $\{\theta\} \cup \{x_i\}$, and suppose we wish to compute $\nabla_\theta \mathcal{G}(\theta, x_1, \ldots, x_n)$. We would like to be able to compute the gradient of any node with respect to any of its inputs, as this would allow us to use the well-known backpropagation algorithm to compute $\nabla_\theta \mathcal{G}$. Unfortunately, it is often impossible to write the gradient of an expectation in closed form; we shall instead estimate $\nabla_\theta \mathcal{G}$ given a sample from the stochastic nodes of $\mathcal{G}$.

Suppose $\mathcal{G}$ has a output node $\mathbb{E}_z[f]$, the only expectation node in $\mathcal{G}$. Suppose moreover that $\mathbb{E}_z[f]$ has inputs $\{\xi_i\}$ (apart from f) so $z \sim p(\cdot, \xi_1 \ldots \xi_m)$, and suppose f has inputs $\{y_i\}$. Note that to satisfy the definition of a nested stochastic computation graph f must ultimately depend on z, so we write f as $f(y_1, \ldots, y_l; z)$. See Figure S9 for a visual representation of $\mathcal{G}$.

If we wish to compute $\nabla_\omega \mathbb{E}_z\left[f(y_1, \ldots, y_l; z)\right]$, using the likelihood ratio interpretation of the score function [6] and given a sample $\{z_i\}_{1 \le i \le N}$ of z, we can write

$$\hat{L}(\omega) = \frac{1}{N} \sum_i f(y_1, \ldots, y_l; z) \mathcal{L}(z_i), \tag{12}$$

where $\mathcal{L}$ is the likelihood ratio given by

$$\mathcal{L}(z_i) = \frac{p(z_i; \xi_1, \ldots, \xi_m)}{p(z_i; \xi'_1, \ldots, \xi'_m)}, \tag{13}$$

and setting $\xi'_i = \xi_i$ gives

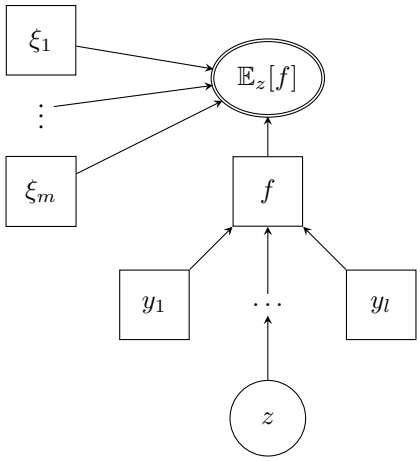


Figure S9: Nested stochastic computation graph with a single expectation node.

$$\nabla_\omega \mathcal{L}(z_i) = \frac{\nabla_\omega p(z_i; \xi_1, \ldots, \xi_m)}{p(z_i; \xi_1, \ldots, \xi_m)} \tag{14}$$

$$= \nabla_\omega \log p(z_i; \xi_1, \ldots, \xi_m). \tag{15}$$

Note that f can either depend on ω directly, if $\omega \in \{y_i\}$, or through the distribution of z, if $\omega \in \{\xi_i\}$. Differentiating, we have

$$\nabla_\omega \hat{L}(\omega) = \frac{1}{N} \sum_i f(y_1, \ldots, y_l; z) \nabla_\omega \mathcal{L}(z_i) + \nabla_\omega f(y_1, \ldots, y_l; z) \mathcal{L}(z_i), \tag{16}$$

and setting $\xi_i' = \xi_i$ we see that $\nabla_\omega \hat{L}(\omega)$ is an estimate of $\nabla_\omega \mathbb{E}_z \left[f(y_1, \ldots, y_l; z) \right]$.

Generalizing this trick to an arbitrary nested stochastic computation graph $\mathcal{G}$, we see that creating a surrogate loss function $\hat{L}$ is as simple as replacing each expectation node with a sample mean as in Equation 12, weighted by the likelihood ratio $\mathcal{L}(z_i)$. Note that since $\mathcal{L}(z_i) = 1$, the surrogate loss is simply the method of moments estimate of the true loss.

Considering again the graph of Figure S8, we can construct a surrogate loss function

$$\hat{L}(\theta, x_0) = \frac{1}{N} \sum_i \left(f_1(x_1^i) + \sum_j f_2(x_2^i) \mathcal{L}(x_2^i; \theta) \right) \mathcal{L}(x_1^i; \theta, x_0). \tag{17}$$

While this may not seem like very much of an improvement at first, it is insightful to note how similar the forms of Equations 10 and 17 are. In particular, this similarity makes it straightforward to write a custom "expectation" operation for use in automatic differentiation software which computes the sample mean weighted by the likelihood ratio.

S5 Stochastic Computation Graphs for ES and Evolvability ES

As mentioned in the main text of the paper, we can estimate the gradients of the ES and Evolvability ES loss functions with the score function estimator because we can represent these loss functions as (nested) stochastic computation graphs. Figure S10 shows a (nested) stochastic computation graph representing the Standard ES loss function. This yields the following surrogate loss function for ES:

$$\hat{L}(\theta) = \frac{1}{N} \sum_i f(z_i) \mathcal{L}(z_i), \tag{18}$$

where $\mathcal{L}(z_i)$ is the likelihood function.

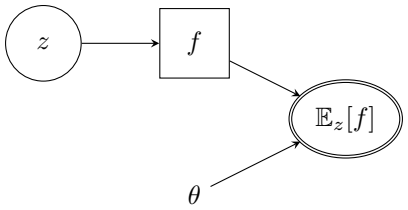


Figure S10: Nested stochastic computation graph representing Natural Evolution Strategies.

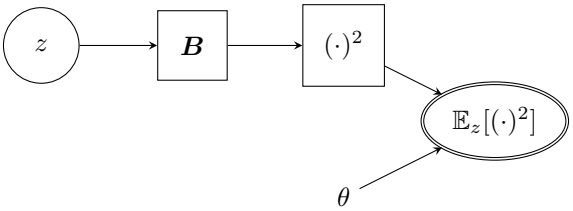


Figure S11: Nested stochastic computation graph representing the loss function of MaxVar Evolvability ES.

Figure S11 shows a nested stochastic computation graph representing MaxVar Evolvability ES, yielding the following surrogate loss function:

$$\hat{L}(\theta) = \frac{1}{N} \sum_i \boldsymbol{B}(z_i)^2 \mathcal{L}(z_i; \theta) \tag{19}$$

Finally, Figure S12 shows a nested stochastic computation graph representing the loss function for MaxEnt Evolvability ES, yielding the following surrogate loss function:

$$\hat{L}(\theta) = -\frac{1}{N} \sum_i \log \left(\sum_j \varphi(\boldsymbol{B}(z') - z) \mathcal{L}(z_j; \theta) \right) \mathcal{L}(z_i; \theta) \tag{20}$$

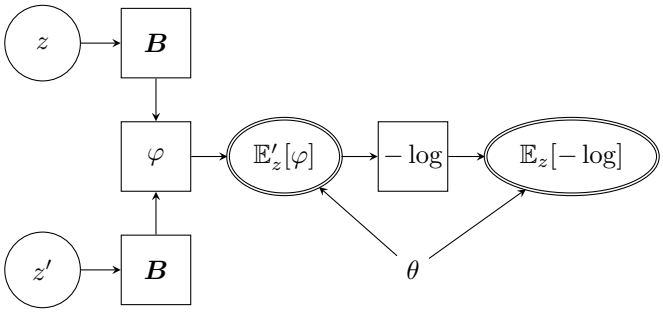


Figure S12: Nested stochastic computation graph representing the loss function of MaxEnt Evolvability ES.