Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients

Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley
Uber AI Labs, San Francisco, CA
{joel.lehman,jayc,jeffclune,kstanley}@uber.com

ABSTRACT

While *neuroevolution* (evolving neural networks) has been successful across a variety of domains from reinforcement learning, to artificial life, to evolutionary robotics, it is rarely applied to large, deep neural networks. A central reason is that while random mutation generally works in low dimensions, a random perturbation of thousands or millions of weights will likely break existing functionality. This paper proposes a solution: a family of *safe mutation* (SM) operators that facilitate exploration without dramatically altering network behavior or requiring additional interaction with the environment. The most effective SM variant scales the degree of mutation of each individual weight according to the sensitivity of the network's outputs to that weight, which requires computing the gradient of outputs with respect to the weights (instead of the gradient of error, as in conventional deep learning). This *safe mutation through gradients* (SM-G) operator dramatically increases the ability of a simple genetic algorithm-based neuroevolution method to find solutions in high-dimensional domains that require deep and/or recurrent neural networks, including domains that require processing raw pixels. By improving our ability to evolve deep neural networks, this new safer approach to mutation expands the scope of domains amenable to neuroevolution.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; *Neural networks*; *Artificial life*; *Evolutionary robotics*;

KEYWORDS

Neuroevolution, mutation, deep learning, recurrent networks

ACM Reference Format:
Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. 2018. Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '18).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205473

1 INTRODUCTION

Neuroevolution (NE; [34]) combines evolutionary algorithms (EAs) and artificial neural networks (NNs). It has long been popular when gradient information with respect to the task is unavailable or difficult to obtain, e.g. in artificial life [11] and evolutionary robotics [20]. Interestingly for NE, recent advances in deep learning have demonstrated the value of large NNs [5], which could seemingly open up new horizons for NE if only it could scale to effectively evolve NNs with millions of parameters (although recent work suggests this may sometimes already be possible [22, 25]). However, NE historically has operated on much smaller networks, most often on the order of tens or hundreds of parameters.

The challenge is that large NNs induce a tension between how fast evolution can theoretically proceed and the degree of weight perturbations applied as a mutation operator. If few weights are perturbed in each generation, it would take many generations for all weights to be tuned; but if many weights are perturbed, changes to the NN's functionality may be too drastic for search to proceed systematically. Indeed, such concerns inspired research into indirect encodings [29], wherein a compact genotype is expanded at evaluation-time into a larger NN phenotype. While such indirect encodings have enabled evolving large NNs [27, 31], even the indirect encoding itself can become high-dimensional and thereby complicate evolution. A further challenge is that in deep and recurrent NNs, there may be drastic differences in the *sensitivity* of parameters; for example, perturbing a weight in the layer immediately following the inputs has rippling consequences as activation propagates through subsequent layers, unlike perturbation in layers nearer to the output. In other words, simple mutation schemes are unlikely to tune individual parameters according to their sensitivity.

To address these challenges with mutating large and deep NNs, this paper pursues the largely unexplored approach of considering perturbation in the space of an NN's *outputs* rather than only in the space of its *parameters*. By considering the NN's structure and the context of past inputs and outputs, it becomes possible to deliberately construct perturbations that avoid wrecking functionality while still encouraging exploring new behaviors. The aim is to ensure that an offspring's NN response will not diverge too drastically from the response of its parent. For example, when the NN is differentiable (which does not require that the *task* or *reward* is differentiable), gradient information can estimate how sensitive the NN's output is to perturbations of *individual* parameters. The opportunity to exploit gradients in *mutation* instead of error or reward, as in stochastic gradient descent (SGD), is an intriguing hint that the line between deep learning and neuroevolution is more blurry than may previously have been appreciated.

This insight leads to two approaches to generating safer NN mutations. One is called *safe mutation through rescaling* (SM-R): At the expense of several NN forward passes, a line search can rescale the magnitude of a raw weight perturbation until it is deemed safe, which does not require the NN to be differentiable. The second is called *safe mutation through gradients* (SM-G): When the NN is differentiable, the sensitivity of the NN to relevant input patterns can

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205473

be calculated (at the expense of a backward pass). Importantly, the assumption underlying these approaches is that *domain* evaluation (i.e. rollouts) is expensive relative to *NN* evaluation (i.e. forward or backward NN propagation). Interestingly, both approaches relate to effective mechanisms from deep learning, e.g. adaptive-learning rate methods [8] or trust regions [26], although here there is distinct motivation and setting (i.e. SM-R and SM-G generate pure variation independent of reward, unlike such deep learning methods).

Both approaches are tested in this paper against a suite of supervised and reinforcement learning (RL) tasks, using large, deep, and recurrent NNs. Note that the EA in these experiments is a simple steady-state algorithm on high-dimensional vectors, so there is no additional sophisticated machinery at work. The tasks build in complexity, ultimately showing their promise for scaling NE to challenging and high-dimensional domains, such as learning to play video games from raw pixels. The results show that in some domains where it would otherwise be be impossible to effectively evolve networks with hundreds of thousands of connections or over a dozen layers (at least with a traditional EA), SM-G entirely changes what is possible and with little computational cost. Furthermore, the value of gradient information is confirmed by an evident advantage of SM-G over SM-R when gradient information is available, especially in evolving large recurrent networks. The conclusion is that SM-G may help to unlock the power of large-scale NN models for NE and EAs in general.

2 BACKGROUND

Next, sensitivity-aware deep learning, deep NE, and sensitivity-aware mutations for evolutionary computation (EC) are reviewed.

2.1 Sensitivity-aware Deep Learning

There is awareness in deep learning that parameter sensitivity is important. For example, adaptive learning-rate optimizers like ADAM [8] in effect take smaller steps when the error gradient with respect to a particular parameter is large (i.e. it is a sensitive parameter), and larger steps when the error gradient is small (i.e. the parameter is relatively insensitive). The SM-G method proposed here can be seen as having motivation similar to ADAM, but with respect to generating *variation* in an NN's outputs instead of directly *reducing* error. Trust regions in policy gradient methods [26] also bear similarity to SM-R and SM-G, in that they attempt to maximize improvement to an NN while limiting the degree of functional change to its behavior. A key difference is that SM-R and SM-G are uninformed by performance, and adapt candidate perturbations solely to attain a desired amount of NN output change. Additionally, other families of deep learning enhancements can be seen as attempting to reduce or normalize sensitivity. For example, long short-term memory (LSTM; [7]) units are designed to avoid some of the instability of vanilla recurrent neural networks (RNNs). SM-R and SM-G can be seen as complementary to such architectural changes, which may also enable more consistent mutation.

2.2 Deep Neuroevolution

Recently there has been increased interest in NE from deep learning researchers evolving the architecture of deep NNs [1, 17], which otherwise requires domain knowledge to engineer. This setting is a natural intersection between EC and deep learning; evolution discovers the structure of a deep network (for which gradient information is unavailable), while deep learning tunes its parameters through SGD. The aim in this paper is instead to exploit gradients to inform variation *within* NE, meaning that this technique can be applied to problems difficult to formulate for deep learning (e.g. open-ended evolution or divergent search), or can improve upon where NE is already competitive with deep learning (e.g. the Deep GA of Petroski Such et al. [22]).

Also recently, Salimans et al. [25] demonstrated that with access to large-scale computation a form of evolution strategy (ES) scales surprisingly well to evolve deep NNs, although it remains unclear how to generalize such an approach to EAs as a whole, or when subject to a computational budget. Interestingly, this form of ES implicitly results in a limited form of safe mutation itself [12]. The approach here aims to be less computationally expensive, and to generalize across EAs; results such as Koutník et al. [10] and Petroski Such et al. [22] demonstrate the promise of such general deep NE, which safe mutation could potentially further catalyze. Some previous work in indirect encoding of NNs also are forms of deep NE [27, 31]. However, indirect encoding offers its own challenges and thus it is useful to have direct encoding approaches to deep NE, such as SM-R and SM-G. Additionally, indirect encodings may equally benefit from the methods proposed here.

2.3 Informed Mutation in EC

Because mutation is a critical EC operator, many approaches target it for improvement. For example, estimation of distribution algorithms (EDAs; [21]) iteratively build and exploit probabilistic models of how promising solutions are distributed; such models can potentially capture the sensitivity of parameters when generating new individuals, although building such models is often expensive in practice. A related approach called natural evolution strategies (NES; [32]) directly optimizes a distribution of solutions. This distributional optimization may indirectly encourage safer mutations by guiding the search to robust areas [12] or by adaptively adjusting the variance of the distribution on a per-parameter basis through domain feedback. Related to NES and EDAs, CMA-ES learns to model pair-wise dependencies among parameters, aimed at generating productive offspring [6]. In contrast, the approaches proposed in this paper do not assume a formal distributional approach and attempt to measure sensitivity without interactions with the domain, allowing the paradigm to generalize to all EAs. Genetic policy optimization [2] is a recent approach that hybridizes EAs and policy gradients; in effect it applies policy gradients as a reward-optimizing variation operator for a specialized small-population EA, using additional domain rollouts and mechanisms such as imitation learning and stochastic-output NNs. (In contrast, the safe mutations in this paper require no additional domain evaluations.)

Interestingly, these mechanisms all attempt to learn from reward, which limits applying them to less-conventional EAs. For example, in EAs focused on creative divergence [13, 19], a population may span many non-stationary modes of high reward, thereby increasing the challenge for approaches that model and track reward distributions. Similarly, within artificial life or open-ended evolution, the concept of an overarching reward function may not even be meaningful. Other EC research explores how *selection pressure* can drive search towards robust or evolvable areas of the search

space [14, 33] or how evolution can produce healthier variation when mutation operators can themselves vary [16]. Such methods may naturally complement SM-G and SM-R, e.g. self-adaptation could adapt the intensity of SM's informed mutations.

3 APPROACH

The general approach for safe mutations in this paper is to choose weight perturbations 扰动 in an informed way such that they produce limited change in the NN's response to representative input signals. The idea is to exploit sources of information that while generally are freely available, are often ignored and discarded. In particular, an archive of representative *experiences* and corresponding NN *responses* can be gathered during an individual's evaluation, which can serve to ground how dramatically a weight perturbation will changes the NN's responses, and thereby inform how its offspring are generated. Secondly, when available, knowledge about the NN structure can also be leveraged to estimate the local effect of weight perturbations on an NN's outputs (as explained later).

To frame the problem, consider a parent individual with parameters $\boldsymbol{w}$ and a potential parameter perturbation vector $\boldsymbol{\delta}$. While in general $\boldsymbol{\delta}$ can be sampled in many ways, here the assumption is that $\boldsymbol{\delta}$ is drawn from an isotropic normal distribution (i.e. the standard deviation is the same for each parameter of the NN), as in Salimans et al. [25]. When the parent is evaluated, assume that a matrix $\boldsymbol{X}_{ij}$ is recorded, consisting of the parent's ith sampled input experience (out of I total sampled experiences) of its jth input neuron, along with the NN's corresponding output response, $Y_{ik} = \text{NN}(\boldsymbol{X}_i; \boldsymbol{w})_k$, to each experience (where k indexes the NN's outputs). Note that $\text{NN}(\boldsymbol{x}; \boldsymbol{w})$ represents the result of forward-propagating the input vector $\boldsymbol{x}$ through an NN with weights $\boldsymbol{w}$.

It is then possible to express how much an NN's response *diverges* as a result of perturbation $\boldsymbol{\delta}$:

$$\text{Divergence}(\boldsymbol{\delta}; \boldsymbol{w}) = \frac{\sum_i \sum_k (\text{NN}(\boldsymbol{X}_i; \boldsymbol{w})_k - \text{NN}(\boldsymbol{X}_i; \boldsymbol{w} + \boldsymbol{\delta})_k)^2}{I}. \quad (1)$$

With this formalism, one can specify the desired amount of divergence (i.e. a mutation that induces some limited change), and then search for or calculate a perturbation $\boldsymbol{\delta}$ that satisfies the requirement. There are different ways to approach safe mutation from this perspective. In particular, the next section explores a simple mechanism to rescale the magnitude of a weight perturbation to encourage safe mutations when the NN is not differentiable. The following section then introduces more flexible perturbation-adjustment methods that exploit gradients through the model. Note that section 1 of the supplemental material describes experiments with a simple poorly-conditioned model that ground intuitions about specific properties of the different SM methods; these experiments are referenced below when relevant and are optional for understanding the paper as a whole.

3.1 Safe Mutation through Rescaling

One approach to satisfying a specific level of divergence in equation 1 is called *safe mutation through rescaling* (SM-R). The idea is to decompose $\boldsymbol{\delta}$ into a *direction* vector in parameter space and a *magnitude* scalar that specifies how far along the direction to perturb: $\boldsymbol{\delta} = \delta_{\text{magnitude}} \boldsymbol{\delta}_{\text{direction}}$. First, the direction is chosen randomly. Then, the scalar $\delta_{\text{magnitude}}$ is optimized with a simple line search to target a specific amount of divergence, which becomes a search hyperparameter replacing the traditional mutation rate.

Importantly, because the parent's experiences have been recorded, this rescaling approach does not require additional domain evaluations, although it does require further NN forward passes (i.e. one for each iteration of the line search, assuming that the sample of experiences is small enough to fit in a single mini-batch). While this approach can achieve variation that is safe by some definition, the effects of mutation may be dominated by a few highly-sensitive parameters (see the Easy and Medium tasks in supplemental material section 1 for a toy example); in other words, this method can rescale the perturbation as a whole, but it cannot granularly rescale each dimension of a perturbation to ensure it has equal opportunity to be explored. The next section describes how gradient information can be exploited to adjust not only the magnitude, but also to reshape the direction of perturbation as well.

3.2 Safe Mutation through Gradients

A more flexible way to generate safe variation is called *safe mutation through gradients* (SM-G). The idea is that if the NN targeted by SM is *differentiable*, then gradient information can *approximate* how an NN's outputs vary as its weights are changed. In particular, the output Y_{ik} of the NN can be modeled as a function of weight perturbation $\boldsymbol{\delta}$ through the following first-order Taylor expansion:

$$Y_{ik}(\boldsymbol{X_i}, \boldsymbol{\delta}; \boldsymbol{w}) = NN(\boldsymbol{X}_i; \boldsymbol{w})_k + \boldsymbol{\delta} \nabla_{\boldsymbol{w}} NN(\boldsymbol{X}_i; \boldsymbol{w})_k$$

This approximation illustrates that the magnitude of each output's gradient with respect to any weight serves as a local estimate of the sensitivity of that output to that weight: It represents the *slope* of the NN's response from perturbations of that weight. By summing such per-output weight sensitivities over outputs, the result is the overall sensitivity of a particular weight. More formally, sensitivity vector $\boldsymbol{s}$ containing the sensitivities of all weights in a NN can be calculated as:

$$\boldsymbol{s} = \sqrt{\sum_k \left(\frac{\sum_i \text{abs}(\nabla_{\boldsymbol{w}} NN(\boldsymbol{X}_i)_k)}{I} \right)^2}.$$

One simple approach to adjust a perturbation on a per-parameter basis is thus to normalize a perturbation by this sensitivity:

$$\boldsymbol{\delta}_{\text{adjusted}} = \frac{\boldsymbol{\delta}}{\boldsymbol{s}}.$$

Note that calculating $\boldsymbol{s}$ in practice requires taking the average absolute value of the gradient over a batch of data (the absolute value reflects that we care about the magnitude of the slope and not its sign); unfortunately this cannot be efficiently calculated within popular tensor-based machine learning platforms (e.g. TensorFlow or PyTorch), which are optimized to compute gradients of an aggregate scalar (e.g. average loss over many examples) and not aggregations over functions of gradients (e.g. summing the absolute value of per-example gradients). To compute this *absolute gradient* variant of SM-G (*SM-G-ABS*) requires a forward and backward pass for *each* of the parent's experiences, which is expensive. A less-precise but more-efficient approximation is to drop the absolute value function:

$$\boldsymbol{s}_{\text{SUM}} \approx \sqrt{\sum_k \left(\sum_i \nabla_{\boldsymbol{w}} NN(\boldsymbol{X}_i)_k \right)^2},$$

which is referred to as the *summed gradient* variant, or *SM-G-SUM*. Note that each output gradient is no longer averaged over all I timesteps of experience, which empirically improves performance, potentially by counteracting the washout effect from summing together potentially opposite-signed gradients. That is, if modifying a weight causes positive and negative output changes in different experiences, the measured sensitivity of that weight would be *washed-out* in the sum of these opposite-signed changes (see the Gradient Washout task in supplemental material section 1 for a toy example); later experiments explore whether such dilution imposes a cost in practice.

A final SM-G approach is to consider gradients of the divergence equation (equation 1) itself, i.e. how does perturbing a weight affect the formalism this paper adopts to define safe mutations in the first place. However, a *second-order* approximation must be used, because the gradient of equation 1 with respect to the weights is uniformly zero when evaluated at the NN's current weights ($\boldsymbol{\delta} = 0$), i.e. divergence is naturally a global minimum because the comparison is between two NNs with identical weights. With second-order information evaluated at $\boldsymbol{\delta} = 0$, the gradient of the divergence as weights are perturbed along a randomly-chosen perturbation direction $\boldsymbol{\delta_0}$ can be approximated as:

$$\begin{aligned}\nabla_{\boldsymbol{w}}(\text{Divergence}(0+\boldsymbol{\delta}_0;\boldsymbol{w})) &\approx \nabla_{\boldsymbol{w}}\text{Divergence}(0;\boldsymbol{w}) + \\ &\quad H_{\boldsymbol{w}}(\text{Divergence}(0;\boldsymbol{w}))\boldsymbol{\delta_0} \\ &\approx H_{\boldsymbol{w}}(\text{Divergence}(0;\boldsymbol{w}))\boldsymbol{\delta_0}, \quad (2)\end{aligned}$$

where H is the Hessian of Divergence with respect to $\boldsymbol{w}$. While calculating the full Hessian matrix of a large NN would be prohibitively expensive, the calculation of a Hessian-vector product (i.e. the final form of equation 2) imposes only the cost of an additional backwards pass; the insight is that the full Hessian is not necessary because what is important is the curvature in only a single particular random direction in weight space $\boldsymbol{\delta}$. Given this estimate of the gradient in the direction of the mutation $\boldsymbol{\delta}$, per-weight sensitivity for this second-order SM-G (i.e. *SM-G-SO*) can then be calculated in a similar way to SM-G-ABS:

$$\boldsymbol{s}_{\text{SO}} = \sqrt{\text{abs}(H_{\boldsymbol{w}}(\text{Divergence}(0;\boldsymbol{w}))\boldsymbol{\delta})},$$

and the perturbation $\boldsymbol{\delta}$ can similarly be adjusted by dividing by the weight sensitivity vector $\boldsymbol{s}_{\text{SO}}$.

4 SIMPLE NEUROEVOLUTION ALGORITHM

Where not otherwise noted, the experiments in this paper all apply the same underlying NE algorithm, which is based on a simple steady-state EA (i.e. there are no discrete generations and individuals are instead replaced incrementally) with standard tournament selection with size 5. All mutation operators build upon a simple control mutation method based on the successful deep-learning ES of Salimans et al. [25], where the entire parameter vector is perturbed with fixed-variance Gaussian noise. Each NN weight vector composing the initial population is separately randomly initialized with the Xavier initialization rule [3], which showed promise in preliminary experiments. For simplicity the algorithm does not include crossover, although there may also be interesting SM-inspired approaches to safe crossover. Evolution proceeds until a solution is evolved or until a fixed budget of evaluations is exhausted. Source code for the NE algorithm and SM operators is available from: https://github.com/uber-common/safemutations.

The strength of mutations is tuned independently with a grid search for each method on each domain (including the control). For methods besides SM-R such strength corresponds to the variance of the Gaussian weight-vector perturbations, while for SM-R the severity of mutation is varied through adjusting the targeted amount of divergence in equation 1. Specific mutation strength settings are noted in the supplemental material.

5 EXPERIMENTS

This section explores the scalability of SM techniques by applying them to domains of increasing difficulty, culminating in first-person traversal of a rendered 3D environment from raw pixels.

5.1 Recurrent Parity Task

To highlight a class of NN models for which SM-G might provide natural benefit, this section introduces a simple recurrent classification task, called *Recurrent Parity*. In this task an NN must classify the parity of a bit-sequence (i.e. whether the sequence contains an odd number of ones) presented to it sequentially in time. Recurrent networks are known to exhibit vanishing and exploding gradients [7], which from a variation point of view would manifest as weights with tiny and massive sensitivities, respectively.

A two-layer RNN network with one input, two recurrent hidden layers of 20 nodes each, and one output is trained to memorize all sixteen 4-bit parities. This network has approximately 1,300 parameters. Twenty independent evolutionary runs are conducted for each method for a maximum of 100,000 evaluations each. Specific hyperparameters for each method are noted in the supplemental material.

Figure 1 shows the fraction of successful runs across evaluations for each run. All SM-G methods evolve solutions significantly faster than the control (Mann-Whitney U-test; $p < 0.001$); SM-R's solution time is significantly better than the control only if failed runs of the control are included in the calculation ($p < 0.05$). To support the intuition motivating the SM-G family, i.e. that SM-G mutations are safer than control mutations, all solutions *evolved* by each mutation method (20 for both SM-G methods and 17 for Control; 3 Control runs did not solve the task) are subject to 50 *post-hoc* perturbations from each mutation method. The robustness of each individual solution/post-hoc-mutation combination is calculated as the average fraction of performance retained after perturbation. The result is that SM-G methods result in significantly more robust mutations no matter what mutation method generates the solution (Mann-Whitney U-test; $p < 0.001$); more details are shown in supplemental material figure 2.

5.2 Breadcrumb Hard Maze

The purpose of this experiment is to explore whether SM approaches have promise for evolving deep networks in an RL context. The Breadcrumb Hard Maze (shown in figure 2) was chosen as a representative low-dimensional continuous control task, and is derived from the Hard Maze benchmark of Lehman and Stanley [13]. An evolved NN controls a wheeled robot embedded within a maze (figure 2a), with the objective of navigating to its terminus. The

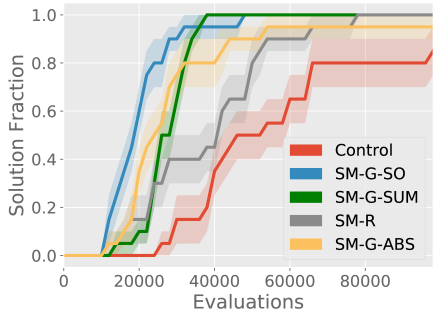


Figure 1: Performance on the Recurrent Parity task across methods. The plot shows the fraction of solutions evolved by each method across evaluations, for twenty independent runs of each method. Each SM-G approach solves the task significantly faster than the control, highlighting the potential for SM to enhance evolution of recurrent NNs.

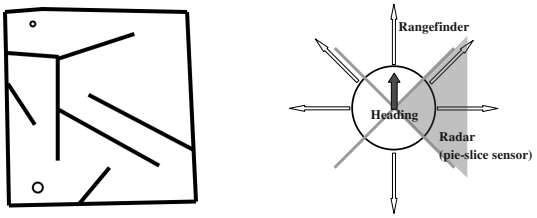
robot receives egocentric sensor information (figure 2b) and has two effectors that control its velocity and orientation, respectively.

In its original instantiation, the hard maze was intended to highlight the role of deception in search, and fitness was intended to be a poor compass. Because this work focuses on a different issue (i.e. the scalability of evolution to deep networks), the fitness function should instead serve as a reliable measure of progress. Thus, fitness in this breadcrumb version of the Hard Maze domain is rewarded as the negation of the A-star distance to the maze's goal from the robot's location at the end of its evaluation, i.e. fitness increases as the navigator progresses further along the solution path (like a breadcrumb trail). Note that a similar domain is applied for similar reasons in Risi and Stanley [24].

Past work applied NEAT to this domain, evolving small and relatively shallow NNs [13, 24]. In contrast, to explore scaling to deep NNs where mutation is likely to become brittle, the NN applied here consists of 16 feed forward hidden layers of 8 units each, for a total of 1,266 evolvable parameters. The activation function in hidden layers is the SELU [9], while the output layer has unsigned sigmoid units. The specific hyperparameters for the NE algorithm and mutation operators are listed in the supplemental material.

5.2.1 Results. Figure 3 shows results across different mutation approaches. SM-G-SO evolves solutions significantly more quickly than the control (Mann-Whitney U-test; $p < 0.005$). The only method to solve the task in all runs was SM-G-ABS, and although the difference in number of solutions did not differ significantly from the control at the end of evolution (Barnard's exact test; $p > 0.05$), at 60,000 evaluations it had evolved significantly more solutions than the control and SM-R (Barnard's exact test; $p < 0.05$). SM-R performs poorly in this domain, suggesting that nuanced gradient information may often provided greater benefit to crafting safe variation; similarly, SM-G-SUM performs no differently from the control, suggesting the gradient washout effect may affect this domain. A video in the supplemental material highlights the qualitative benefits of the SM approaches when mutating solutions.

5.2.2 Large-scale NN Experiment. Additional experiments in this domain explore the ability of SM-G methods to evolve NNs with up to a hundred layers or a million parameters; before now neuroevolution of NNs with over 10 layers has little precedent.



(a) Hard Maze Map **(b) Maze Navigating Robot**

Figure 2: The Breadcrumb Hard Maze domain. The maze's layout is shown in (a), while the maze navigating robot and its sensors is shown in (b). In (a), the large circle represents the robot's starting position and the small circle represents the goal. In (b), each arrow outside of the robot's body is a rangefinder sensor measuring the distance to the closest obstacle in that direction. The robot has four pie-slice sensors that act as a compass, activating when a line from the goal to the center of the robot falls within the pie-slice (i.e. irrespective of intervening walls). The solid arrow indicates the robot's heading. Navigating robots are rewarded for ending in locations with low A-star distance to the goal.

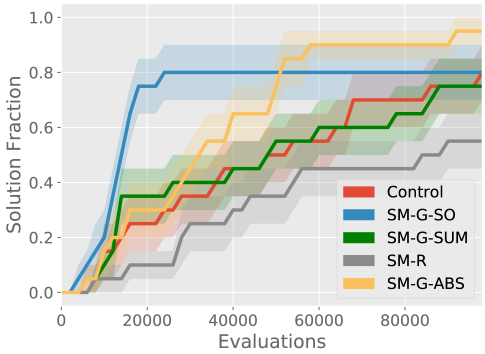


Figure 3: Performance on the Breadcrumb Hard Maze across methods. The fraction of solutions evolved by each method is shown over increasing evaluations. SM-G-SO evolves solutions significantly more quickly than the standard mutation control, and only SM-G-ABS evolves solutions in each of its 20 independent runs. Interestingly, SM-G-SUM's performance mirrors the control, while SM-R under-performs the other methods. The conclusion is that some domains benefit from SM methods that exploit more principled gradient information (SM-G-ABS and SM-G-SO).

Three network architectures inspired by wide residual networks [35] were tested, consisting of 32, 64, and 101 Tanh layers, with residual skip-ahead connections every four layers. The 32 and 64-layer models are designed to explore parameter-size scalability, and have 125 units in each hidden layer, resulting in models with approximately half a million and a million parameters, respectively. The 101-layer model is designed instead to explore scaling NE to extreme depth; each layer contains fewer units (48 vs. 125) than the 32 and 64-layer models, resulting in fewer total parameters (approximately 200,000) despite the NN's increased depth. While the previous experiment shows that such capacity (a million parameters

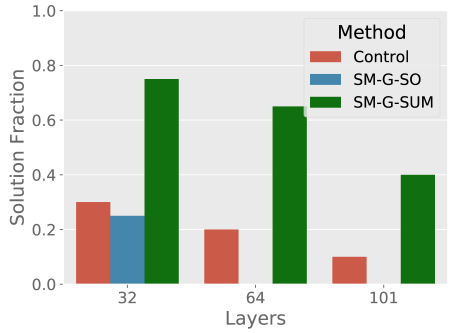


Figure 4: Performance on the large-scale NN task across methods. The fraction of solutions successfully evolved by each method over 20 runs is shown. Although performance degrades with increasing layers for all methods, SM-G-SUM evolves significantly more solutions than the standard mutation control and SM-G-SO in each of the 32-layer, 64-layer, and 101-layer models. The conclusion is that SM-G can help to unlock the potential of NNs with up to a million parameters or a hundred layers.

or 100 layers) is unnecessary to solve this task, success nonetheless highlights the potential for NE to scale to large models. Note that hyperparameters are fit using grid search on the 32-layer model, and are then also applied to the 64 and 101-layer models.

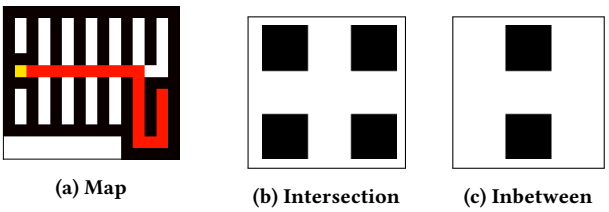
Figure 4 shows results across different mutation methods for each of the three architectures, with 20 independent runs for each combination of model and method. In each case, SM-G-SUM evolves significantly more solutions than either SM-G-SO or the control ($p < 0.05$; Barnard's exact test). The conclusion is that SM-G shows potential for effectively evolving large-scale NNs. Note that more detailed training plots are included in the supplemental material.

5.3 Topdown Maze

The Topdown Maze domain is designed to explore whether safe mutation can accelerate the evolution of deep recurrent convolutional NNs that learn from raw pixels. The motivation is that this is a powerful architecture that enables learning abstract representations without feature engineering as well as integrating information over time; similar combinations of recurrence and convolution have shown considerable promise in deep RL [18].

In this domain, the agent receives a visual 64x64 input containing a local view of a maze (i.e. it cannot see the whole maze at once) as a grayscale image, and has discrete actions that navigate one block in each of the four cardinal directions. Because the maze (figure 5a) has many identical intersections, which the agent cannot distinguish by local visual information alone (figures 5b and c), solving the task requires use of recurrent memory.

These experiments focus on comparing the more computationally-scalable variants of SM-G (i.e. SM-G-SUM and SM-G-SO) to the control mutation method, because of the complexity and size of the RNN. Note that because this NN is recurrent, backpropagation through time is used for the SM-G approaches when calculating weight sensitivity, i.e. weight sensitivity in SM-G is informed by the cascading effects of signals over time.



(a) Map (b) Intersection (c) Inbetween

Figure 5: Topdown Maze domain. The (a) 2D grid-world maze is shown in which the agent is embedded. A black square indicates a wall and the red path indicates the target trajectory of the agent. One fitness point is awarded for each square along the trajectory the agent touches. Note that the red trajectory is not visible to the agent. Because the agent views only a 3×3 block window around its immediate location, many (b) intersections and (c) positions between intersections are conflated. Successful completion of the maze thus requires integrating information over time by making use of recurrence. Note that each block is rendered as a 21×21 square of the NN's input image.

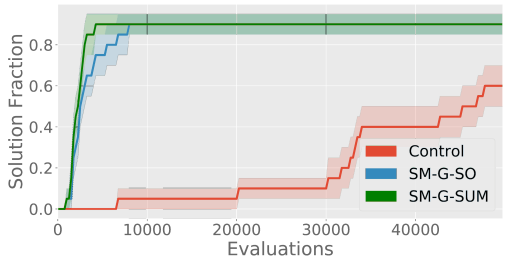


Figure 6: Performance on the Topdown Maze across methods. The fraction of successful independent runs (from the 20 conducted for each method) is shown across SM-G methods and the control mutation method. SM-G-SUM and SM-G-SO solve the task consistently and in relatively few evaluations when compared to the control.

5.3.1 Experimental Settings. The agent receives as input a 64×64 grayscale image and has at most 40 time-steps to navigate the environment. The NN has a deep convolutional core, with two layers of 5×5 convolution with stride 2, to reduce dimensionality, followed by 12 layers of 3×3 convolution with stride 1. All convolutional layers have 12 feature maps and SELU activation. This pathway feeds into an average pooling layer that leads into a two-layer LSTM [7] recurrent network with 20 units each; the signal then feeds into an output layer with sigmoid units. The NN has in total 25,805 evolvable parameters and 17 trainable layers. EA and mutation hyperparameters are provided in the supplemental material.

5.3.2 Results. Figure 6 shows the results in this domain. Both SM-G-SUM and SM-G-SO consistently evolve solutions, and even when the control is successful it requires significantly more evaluations for it to discover a solution (Mann-Whitney U-test; $p < 0.001$). Complementing the results of the simple RNN classification task in section 5.1, the conclusion from this experiment is that the tested SM-G approaches can accelerate evolution of successful memory-informed navigation behaviors.

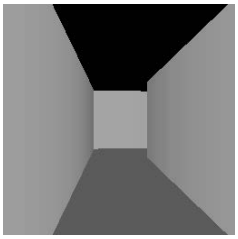


Figure 7: First-person 3D Maze Domain. An agent traverses a 3D first-person environment and is rewarded the further it progresses along the correct path.

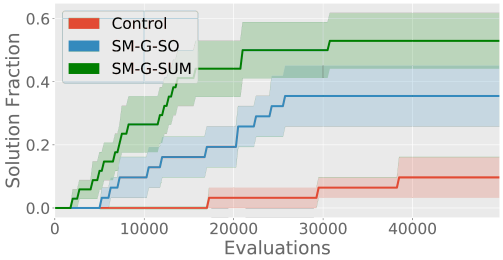


Figure 8: Performance on the First-person 3D Maze across methods. The fraction of successful independent runs (from the 30 conducted for each method) is shown across SM-G methods and the control mutation method. SM-G-SUM and SM-G-SO both solve the task significantly more frequently than does the control.

5.4 First-person 3D Maze

The First-person 3D Maze is a challenge domain in which a NN learns to navigate an environment from first-person 3D-rendered images (figure 7). The maze has the same layout as the Topdown Maze. However, navigation is egocentric and continuous in space and heading, i.e. the agent does not advance block-wise in cardinal directions, but its four discrete actions incrementally turn the agent left or right, or advance or reverse. The agent is given 400 frames to navigate the maze. Note that this domain builds upon the RayCast Maze environment of the PyGame learning environment [30].

5.4.1 Experimental Settings. Input to the NN is a grayscaled 64×64 image. The NN has an architecture nearly identical to that of the ANN of the Topdown Maze, i.e. a deep convolutional core (but with 8 instead of 12 layers) feeding into a two-layer LSTM stack (each composed of 20 units), which connects to an output layer with sigmoid units. There are 20,573 parameters in total. The NN executes the same action 4 frames in a row to reduce the computational expense of the domain, which is bottlenecked by forward propagation of the RNN (for the control mutation method) and a combination of forward and backward RNN propagation (for SM-G approaches). The EA settings are the same as in the Topdown maze, but hyperparameters for each mutation method are fit for this domain separately and are described in the supplemental material.

5.4.2 Results. Figure 8 shows the results in this domain. As in the Topdown Maze domain, both SM-G-SUM and SM-G-SO solve the task significantly more often than does the control ($p < 0.05$; Barnard's exact test). The conclusion is that SM-G methods can help scale NE to learn adaptive behaviors from raw pixel information using modern deep learning NN architectures. A video of a solution from SM-G-SUM is included in the supplemental material.

6 DISCUSSION AND CONCLUSION

The results across a variety of domains highlight the general promise of safe mutation operators, in particular when guided by gradient information, which enables evolving deeper and larger recurrent networks. Importantly, because SM methods generate variation without taking performance into account, they can easily apply to artificial life or open-ended evolution domains, where well-defined reward signals may not exist; SM techniques are also agnostic to the stationarity or uni-modality of reward, and can thus be easily applied to divergent search [13, 19] and coevolution [23]. Additionally, SM-G may well-complement differentiable *indirect encodings* where mutations can have outsize impact; for example, in HyperNEAT [27], CPPN mutations could be constrained such that they limit the change of connectivity in the substrate (i.e. target network) or the substrate's output. This safety measure would not preclude systematic changes in weight, only that those systematic changes proceed *slowly*. While both SM-G and SM-R offer alternative routes to safety, it appears from the initial results in this paper that SM-G (and variants thereof) is likely the more robust approach overall.

Another implication of safe mutation is the further opening of NE to deep learning in general. Results like the recent revelation from Salimans et al. [25] that an evolution strategy (ES) can rival more conventional deep RL approaches in domains requiring large or deep networks such as humanoid locomotion and Atari have begun to highlight the role evolution can potentially play in deep learning; interestingly, the ES of Salimans et al. [25] itself has an inherent drive towards a form of safe mutations [12], highlighting the general importance of such mutational safety for deep NE. Some capabilities, such as indirect encoding or searching for architecture as in classic algorithms like NEAT [28], are naturally suited to NE and offer real potential benefits to NN optimization in general. Indeed, combinations of NE with SGD to discover better neural architectures are already appearing [15, 17]. The availability of a safe mutation operator helps to further ease this ongoing transition within the field to much larger and state-of-the-art network architectures. A wide range of possible EAs can benefit, thereby opening the field anew to exploring novel algorithms and ideas.

In principle the increasing availability of parallel computation should benefit NE. After all, evolution is naturally parallel and as processor costs go down, parallelization becomes more affordable. However, if the vast majority of mutations in large or deep NNs are destructive, then the windfall of massive parallelism is severely clipped. In this way, SM-G can play an important role in realizing the potential benefits of big computation for NE in a similar way that innovations such as ReLU activation [4] (among many others) in deep learning have allowed researchers to capitalize on the increasing power of GPUs in passing gradients through deep NNs.

In fact, one lingering disadvantage in NE compared to the rest of deep learning has been the inability to capitalize on explicit gradient information when it is available. Of course, the quality of the gradient obtained can vary – in reinforcement learning for example it is generally only an indirect proxy for the optimal path towards higher performance – which is why sometimes NE can

rival methods powered by SGD [22, 25], but in general it is a useful guidepost heretofore unavailable in NE. That SM-G can now capitalize on gradient information for the purpose of safer mutation is an interesting melding of concepts that previously seemed isolated in their separate paradigms. SM-G is still distinct from the reward-based gradients in deep learning in that it is computing a gradient entirely without reference to how it is expected to impact reward, but it nevertheless capitalizes on the availability of gradient computation to make informed decisions. In some contexts, such as in reinforcement learning where the gradient may even be misleading, SM-G may offer a principled alternative – while we sometimes may not have sufficient information to know *where* to move, we can still explore different directions in parallel as safely as possible. That we can do so without the need for further rollouts (as required by e.g. Gangwani and Peng [2]) is a further appeal of SM-G.

Furthermore, this work opens up future directions for understanding, enhancing, and extending the safe mutation operators introduced here. For example, it is unclear what domain properties predict which SM or SR method will be most effective. Additionally, similarly-motivated safe *crossover* methods could be developed, suggesting there may exist other creative and powerful techniques for exploiting NN structure and gradient information to improve evolutionary variation. Highlighting another interesting future research direction, preliminary experiments explored a version of SM-G that exploited supervised learning to program a NN to take *specific* altered actions in response to particular states (similar in spirit to a random version of policy gradients for exploration); such initial experiments were not successful, but the idea remains intriguing and further investigation of its potential seems merited.

Finally, networks of the depth evolved with SM-G in this paper have never been evolved before with NE, and those with similar amounts of parameters have rarely been evolved [22]; in short, scaling in this way might never have been expected to work. In effect, SM-G has dramatically broadened the applicability of a simple, raw EA across a broad range of domains. The extent to which these implications extend to more sophisticated NE algorithms is a subject for future investigation. At minimum, we hope the result that safe mutation can work will inspire a renewed interest in scaling NE to more challenging domains, and reinvigorate initiatives to invent new algorithms and enhance existing ones, now cushioned by the promise of an inexpensive, safer exploration operator.

ACKNOWLEDGEMENTS

We thank the members of Uber AI Labs, in particular Thomas Miconi and Xingwen Zhang for helpful discussions. We also thank Justin Pinkul, Mike Deats, Cody Yancey, and the entire OpusStack Team inside Uber for providing resources and technical support.

REFERENCES

[1] Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.
[2] Gangwani, T. and Peng, J. (2017). Genetic policy optimization.
[3] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.
[4] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.
[5] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.
[6] Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.
[7] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
[8] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
[9] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*.
[10] Koutník, J., Schmidhuber, J., and Gomez, F. (2014). Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proc. of the 2014 Annual Conf. on Genetic and Evolutionary Computation*, pages 541–548. ACM.
[11] Langton, C. G. (1989). Artificial life.
[12] Lehman, J., Chen, J., Clune, J., and Stanley, K. O. (2017). ES is more than just a traditional finite-difference approximator. *arXiv preprint arXiv:1712.06568*.
[13] Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.
[14] Lehman, J. and Stanley, K. O. (2013). Evolvability is inevitable: Increasing evolvability without the pressure to adapt. *PLoS ONE*, 8(4):e62186.
[15] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.
[16] Meyer-Nieberg, S. and Beyer, H.-G. (2007). Self-adaptation in evolutionary algorithms. *Parameter setting in evolutionary algorithms*, pages 47–75.
[17] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Navruzyan, A., Duffy, N., and Hodjat, B. (2017). Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*.
[18] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.
[19] Mouret, J. and Clune, J. (2015). Illuminating search spaces by mapping elites. *ArXiv e-prints*, abs/1504.04909.
[20] Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MIT Press, Cambridge.
[21] Pelikan, M., Goldberg, D. E., and Lobo, F. G. (2002). A survey of optimization by building and using probabilistic models. *Comp. opt. and apps.*, 21(1):5–20.
[22] Petroski Such, F., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.
[23] Popovici, E., Bucci, A., Wiegand, R. P., and De Jong, E. D. (2012). *Coevolutionary Principles*, pages 987–1033. Springer Berlin Heidelberg, Berlin, Heidelberg.
[24] Risi, S. and Stanley, K. O. (2011). Enhancing es-hyperneat to evolve more complex regular neural networks. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1539–1546. ACM.
[25] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv e-prints*, 1703.03864.
[26] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897.
[27] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.
[28] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127.
[29] Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130.
[30] Tasfi, N. (2016). Pygame learning environment. https://github.com/ntasfi/PyGame-Learning-Environment.
[31] van Steenkiste, S., Koutník, J., Driessens, K., and Schmidhuber, J. (2016). A wavelet-based encoding for neuroevolution. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pages 517–524. ACM.
[32] Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. (2008). Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3381–3387. IEEE.
[33] Wilke, C. O., Wang, J. L., Ofria, C., Lenski, R. E., and Adami, C. (2001). Evolution of digital organisms at high mutation rates leads to survival of the flattest. *Nature*, 412(6844):331–333.
[34] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.
[35] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.