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 94103
{joel.lehman,jayc,jeffclune,kstanley}@uber.com

Abstract

While *neuroevolution* (evolving neural networks) has a successful track record across a variety of domains from reinforcement learning to artificial life, 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 is likely to break existing functionality, providing no learning signal even if some individual weight changes were beneficial. This paper proposes a solution by introducing a family of *safe mutation* (SM) operators that aim within the mutation operator itself to find a degree of change that does not alter network behavior too much, but still facilitates exploration. Importantly, these SM operators do not require any additional interactions with the environment. The most effective SM variant capitalizes on the intriguing opportunity to scale 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 (which tend to be particularly brittle to mutation), 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.

1 Introduction

Neuroevolution (NE; Lehman and Miikkulainen 2013; Floreano et al. 2008; Yao 1999) combines evolutionary algorithms (EAs) and artificial neural networks. It has long been popular when gradient information with respect to the task is unavailable or difficult to obtain, e.g. in artificial life (Bedau et al., 2000; Langton, 1989) and evolutionary robotics (Nolfi and Floreano, 2000; Bongard et al., 2006; Cully et al., 2015). Interestingly for NE, recent advances in deep learning have demonstrated the value of large neural networks (LeCun et al., 2015; Krizhevsky et al., 2012; Goodfellow et al., 2016), which in principle could open up new horizons for NE if only it could scale to effectively evolve neural networks with millions of parameters (although recent work suggests this may sometimes already be possible (Salimans et al., 2017; Petroski Such et al., 2017)). However, NE historically has operated on much smaller networks, often on the order of tens or hundreds of parameters, due to at least the perception of the so-called curse of dimensionality, as well as more limited computational resources than today.

The challenge is that large or deep neural networks (NNs) induce a tension between how fast evolution can theoretically proceed and the degree of weight perturbations applied as a mutation operator. If only a 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 (Stanley and Miikkulainen, 2003; Hornby and Pollack, 2001; Gruau, 1994), wherein a compact genotype is expanded at evaluation-time into a larger NN phenotype. While such indirect encodings have enabled evolving large NNs (Stanley et al., 2009a; Clune et al., 2011; Hausknecht et al., 2014; van Steenkiste et al., 2016), search in such spaces is often challenging exactly because of their representational power, i.e. small genotypic changes can result in large and correlated phenotypic changes. 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 many subsequent layers, while the effect of a perturbation in layers nearer to the output may have less holistic impact. In other words, it is difficult to choose a mutation scheme that tunes individual parameters according to their sensitivity.

To address these challenges with mutating large and deep NNs, the main idea of this paper is the largely unexplored approach of pursuing perturbation in the space of an NN's *outputs* rather than only in the space of its *parameters*. In other words, by considering the NN's structure and the context of its inputs and outputs when it is applied, it may be possible to construct parameter perturbations that are safe enough to avoid wrecking functionality, but still strong enough to explore new behaviors. That is, we want to ensure that on average an offspring's NN response will not diverge too drastically from the response of its parent. For example, a naive initial weight perturbation could be rescaled until its outputs diverge a desired amount from its parent. Beyond holistic rescaling, when the NN is differentiable (note that this does not require that the *task* or *reward* is differentiable), gradient information can be exploited to estimate how sensitive the NN's output is to perturbations of *individual* parameters. The opportunity to exploit gradients in *mutation* (as opposed to stochastic gradient descent, which is directed towards a desired reward or error minimization in the domain) is an intriguing hint that the line between deep learning through gradient descent on the one hand and neuroevolution on the other is more blurry than may previously have been appreciated.

This insight leads to two concrete approaches to generating safer NN mutations. One is called *safe mutation through rescaling* (SM-R): At the expense of several forward passes, a line search can rescale the magnitude of a candidate weight perturbation until it is deemed safe, which can be applied even when the NN is not differentiable. The second is called *safe mutation through gradients* (SM-G): When the NN is differentiable, the sensitivity of the NN in the neighborhood of relevant input patterns can 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. Interestingly, both approaches relate to mechanisms found effective in deep learning, such as adaptive-learning rate methods (Kingma and Ba, 2014; Riedmiller and Braun, 1993) or trust regions in policy gradients (Schulman et al., 2015), although here they manifest through different motivation in a distinct setting (i.e. SM-R and SM-G are pure variation operators and do not take into account reward the way deep learning methods do).

Both approaches are tested in this paper against a suite of supervised and reinforcement learning (RL) tasks, using large, deep, and recurrent NNs. It is important to note that the EA in these experiments is a simple genetic algorithm on high-dimensional vectors, so there is no additional sophisticated machinery at work. (For example, algorithms like the ES of Salimans et al. (2017) may induce their own implicit pressure towards regions of the search space where mutations are safer (Lehman et al., 2017), but the setup in this paper receives no such additional benefits.) The supervised learning tasks help to clearly illuminate the properties of the proposed approaches, while the RL tasks show 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 GA), 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, where significant variance in the sensitivity of the network to varying different weights often exists. The main conclusion is that SM-G may help to unlock the power of large-scale NN models for neuroevolution and evolutionary algorithms in general.

2 Background

The next section reviews treatments of weight sensitivity in deep learning, previous approaches to deep NE, and finally, other evolutionary computation (EC) approaches to creating more-informed mutations.

2.1 Sensitivity-aware Deep Learning

There is awareness in deep learning that sensitivity to perturbation varies across parameters, and that such sensitivity is important. For example, adaptive learning-rate optimizers like ADAM (Kingma and Ba, 2014) or RMSProp (Tieleman and Hinton, 2012) 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 (Schulman et al., 2015) 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 the NN. 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, batch normalization (Ioffe and Szegedy, 2015) aims to normalize activation between layers such that the distribution of activations stays relatively stable across many layers and also across training time; similarly, long short-term memory (LSTM; Hochreiter and Schmidhuber 1997) 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 networks (Miikkulainen et al., 2017; Fernando et al., 2017), 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 stochastic gradient descent (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. (2017)).

Another recent work has demonstrated that with access to large-scale computation a form of evolution strategy can indeed scale to evolve deep NNs (Salimans et al., 2017), 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 (Lehman et al., 2017). The approach here aims to be less computationally expensive, and to generalize across EAs; results such as Koutník et al. (2014) and Petroski Such et al. (2017) 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; for example, in Hausknecht et al. (2014) the CPPN-based indirect encoding in HyperNEAT encodes a multi-layer NN with more than 900,000 weights, and a recent wavelet-based encoding has evolved NNs of comparable size (van Steenkiste et al., 2016). However, indirect encoding offers its own challenges, such as the tension between regularity and exceptions (Clune et al., 2011), 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 other approaches also target it for improvement. For example, estimation of distribution algorithms (EDAs; Pelikan et al. 2002) iteratively build and exploit probabilistic models of how promising solutions are distributed. Such models can result in

implicitly leveraging the sensitivity of parameters when generating new individuals, although building such models is often expensive in practice and may not capture complex interdependencies. A similar vein of research called natural evolution strategies (NES; Wierstra et al. 2008) directly optimizes a distribution of solutions. This distributional optimization may indirectly encourage safer mutations by guiding the search to robust areas (Lehman et al., 2017), and it is also possible to adaptively adjust the variance of the distribution on a per-parameter basis through feedback aggregated from domain evaluations. Related to NES and EDAs, CMA-ES learns to model pair-wise dependencies among parameters, aimed at generating productive offspring (Hansen et al., 2003). 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 (Gangwani and Peng, 2017) 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 rollouts and mechanisms such as imitation learning and stochastic-output NNs. (In contrast, the safe mutations in this paper require no additional rollouts.)

Interestingly, these mechanisms all attempt to learn from the reward signal, which limits their applicability to less-conventional EAs. For example, in EAs focused on creative divergence (Lehman and Stanley, 2011a; Mouret and Clune, 2015), a population might span many non-stationary modes of high reward, thereby increasing the challenge for approaches attempting to model and track reward distributions. Similarly, within artificial life or open-ended evolution, the concept of an overarching reward function may not be meaningful. Other lines of EC research explore how *selection pressure* can drive search towards robust or evolvable areas of the search space (Lehman and Stanley, 2013; Wilke et al., 2001) or allow evolution to modify mutation operators to more effectively produce healthy variation (Lehman and Stanley, 2011b; Meyer-Nieberg and Beyer, 2007). Such methods may naturally complement or hybridize with SM-G and SM-R because they focus on selection, whereas SM-G and SM-R focus on mutation.

3 Approach

The general approach for safe mutations in this paper is to choose weight perturbations in an informed way such that they produce limited changes 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. (2017). 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 sensor, along with the NN's corresponding output response, $\boldsymbol{Y}_{ik} = \text{NN}(\boldsymbol{X}_i; \boldsymbol{w})_k$, 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 change, but one that is not too drastic), 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.

3.1 Safe Mutation through Rescaling

One straightforward 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:

$$\boldsymbol{\delta} = \delta_{magnitude}\boldsymbol{\delta}_{direction}.$$

If the direction is chosen randomly, then the scalar $\delta_{magnitude}$ can be optimized with a simple line search to target a specific amount of divergence, which becomes a search hyper-parameter 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). One drawback of this approach is that while it can achieve variation that is safe by some definition, the effects of mutation may be dominated by a few highly-sensitive parameters; 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 approach to generating 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 $\boldsymbol{Y}_{ik}$ of the NN can be modeled as a function of weight perturbation $\boldsymbol{\delta}$ through the following first-order Taylor expansion:

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

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:

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

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 minibatch (the absolute value reflects that we care about the magnitude of the slope and not its sign); unfortunately this is a difficult quantity to compute efficiently within popular tensor-based machine learning platforms, which generally compute gradients of scalar quantities and not *scalar functions* of 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 calculate:

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

which is referred to as the *summed gradient* variant, or *SM-G-SUM*. Note that each output gradient is no longer averaged over the I timesteps of distinct experience, which empirically improves

performance and can been seen as counteracting the washout effect of summing the signed gradient. The experiments explore whether there is a cost in practice for using such an approximation.

A final SM-G approach (which largely follows the same abstract scheme as SM-G-SUM and SM-G-ABS) is to directly consider gradients of the divergence equation (equation 1) itself, i.e. how does perturbing a weight affect divergence, given that it is 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:

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

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 $\boldsymbol{s}_{\text{SO}}$.

Note that it is also possible to combine SM-G and SM-R; the idea is that SM-G can adjust weight perturbations according to their sensitivity, and SM-R could operate thereafter to fine-tune the actual effect on the NN to correct approximation errors; preliminary experiments have explored this idea, but it is left to future work to demonstrate its promise.

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 with tournament selection. All mutation operators build upon a simple control mutation method based on the successful deep-learning ES of Salimans et al. (2017), 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 (Glorot and Bengio, 2010), 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 will be released at a later date.

The variance of the noise is the single hyperparameter that is tuned for the control and for all SM-G variants. SM-R replaces that hyperparameter with a scalar that is similar in spirit, i.e. the targeted amount of divergence in equation 1 that the line search attempts to match. Specific hyperparameters for each domain are noted in the appendix, including mutation-specific settings for each method, which were fit per domain through an initial grid search.

5 Illustrative Experiments

The first experiments in this paper are designed to illustrate and validate the intuitions motivating SM-G and SM-R, and are conducted in two simple supervised learning tasks.

5.1 Simple Poorly-Conditioned Model

To gain a clearer intuition of the benefits and costs of SM-R and SM-G variants, it is useful to introduce a toy model constructed purposefully with parameters that significantly vary in their sensitivity:

$$\begin{aligned} \boldsymbol{y}_0 &= 100\boldsymbol{w}_0\boldsymbol{x}_0 \\ \boldsymbol{y}_1 &= 0.1\boldsymbol{w}_1\boldsymbol{x}_1 \end{aligned}, \tag{3}$$

where $\boldsymbol{y}$ are the model's outputs as a function of the inputs $\boldsymbol{x}$ and the current weights $\boldsymbol{w}$. Notice that $\boldsymbol{w}_0$ will have 1,000 times the effect on $\boldsymbol{y}_0$ than $\boldsymbol{w}_1$ has on $\boldsymbol{y}_1$. If the scale of expected outputs for $\boldsymbol{y}_0$ and $\boldsymbol{y}_1$ is similar, then an uninformed mutation operator would have difficulty generating variation that equally respects effects on $\boldsymbol{y}_0$ and $\boldsymbol{y}_1$; likely the effect of mutation on the NN will be dominated by the scale of $\boldsymbol{\delta}_0$.

Now, consider three variations of the task. In the first, called the *Easy* task, there is a single input-output pair to memorize, which depends only on $\boldsymbol{w}_1$:

$$\text{input}_0 = 0.0, 1.0 \quad \text{target}_0 = 0.0, 1.0$$

In the second, called the *Medium* task, there is again a single input-output pair to memorize, but solving the task requires tuning $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$, on which mutations have a substantially different effect:

$$\text{input}_0 = 1.0, 1.0 \quad \text{target}_0 = 1.0, 1.0$$

In the last task, called the *Gradient-Washout* task, there are two input-output pairs, designed to highlight a potential failure case of SM-G-SUM:

$$\begin{aligned} \text{input}_0 &= 1.0, 1.0 & \text{target}_0 &= 1.0, 1.0 \\ \text{input}_1 &= -1.0, -1.0 & \text{target}_1 &= -1.0, -1.0 \end{aligned}$$

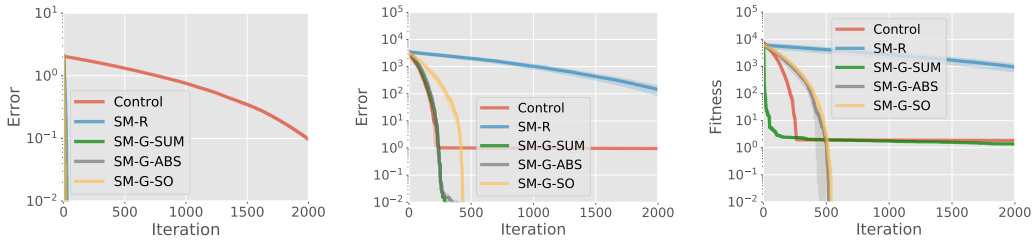
The Easy task is designed to highlight situations in which all SM-G and SM-R variants will succeed, the Medium task highlights when SM-G approaches will have advantage over SM-R, and the Gradient-Washout task highlights situations wherein SM-G-ABS and SM-G-SO have advantage over SM-G-SUM. In particular, in the Gradient-Washout task the only relevant parameter is $\boldsymbol{w}_0$, but due to opposite-sign inputs, the summed gradient of $\boldsymbol{y}_0$ with respect to $\boldsymbol{x}_0$ is 0. The more informative average *absolute value* gradient (used by SM-G-ABS) is 100.

In this experiment, a simple hill-climbing algorithm is applied instead of the NE algorithm described in the previous section. The hill-climber is initialized with small zero-centered noise. Runs last for $2,000$ iterations, wherein an offspring from the current champion is generated, and replaces the champion only if its fitness improves upon that of the champion.

Figure 1 shows the results from 20 independent runs for each mutation method, i.e. control mutation, SM-R, and variations of SM-G. Fitting the motivation of the experiments, the Easy task is solved effectively by all SM variants (which all outperform the control), the Medium task highlights the benefits of SM methods that take advantage of gradient information, and the Gradient-Washout task highlights the benefits of methods that do not sum NN outputs over experiences before calculating sensitivity. Note that with a tuned mutation rate, the control can more quickly solve the Easy task, but the point is to highlight that SM-G methods can normalize mutation by their effect on the output of a model, identifying automatically when a parameter is less sensitive (e.g. w_1 in this task) and can thus safely be mutated more severely.

5.2 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)



(a) Easy Task (b) Medium Task (c) Gradient-Washout Task

Figure 1: **Performance of SM-R and SM-G on the Simple Poorly-conditioned Tasks.** All SM-G and SM-R methods perform well on the (a) Easy task, while the (b) Medium task stymies SM-R, and the (c) Gradient-Washout task highlights the benefits of SM-G-ABS and SM-G-SO.

presented to it sequentially in time. Recurrent networks are known to exhibit vanishing and exploding gradients (Pascanu et al., 2013), 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 appendix.

Figure 2 shows a plot of 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 the failed runs of the control are included in the calculation ($p < 0.05$). Figure 3a supports the intuition behind the SM-G family by perturbing a representative solution (i.e. a NN with perfect fitness) from each method by its own mutation operator; SM-G creates a much greater proportion of high-performing mutations than does the control. To validate this intuition more rigorously, 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. Figure 3b summarizes the results, showing that SM-G methods result in significantly more robust mutations no matter what mutation method generates the solution (Mann-Whitney U-test; $p < 0.001$).

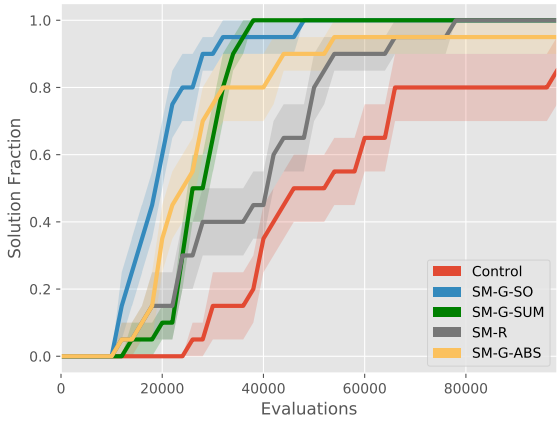
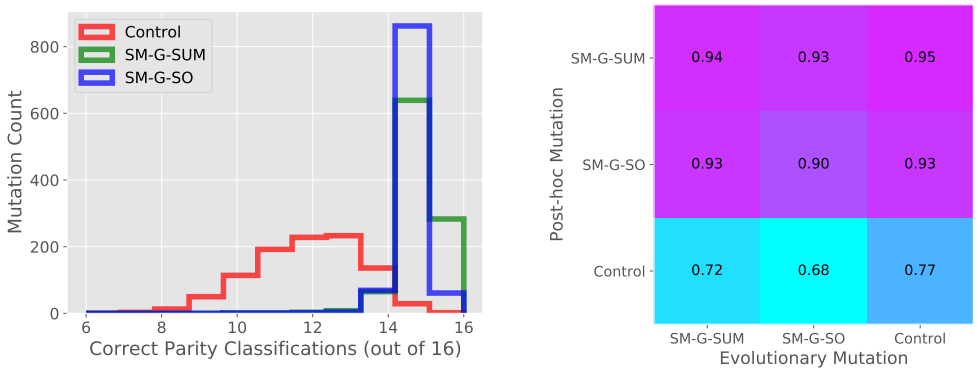


Figure 2: **Performance on the Recurrent Parity task across methods.** The plot shows the fraction of solutions evolved by each method across evaluations, for twenty independent runs of each method. Each SM-G approach solves the task significantly faster than the control, highlighting the potential for SM to enhance evolution of recurrent NNs.



(a) Representative Histogram

(b) Comprehensive Comparison

Figure 3: **Analyzing robustness of solutions in the Recurrent Parity task.** (a) For each mutation method, 1,000 perturbations by that method are generated from a representative solution (evolved by that method). The histogram of correct parity classifications (out of sixteen) achieved by perturbations are shown for SM-G-SUM, SM-G-SO, and the control. (b) The plot shows robustness to mutations averaged across solutions. For each evolutionary mutation method (i.e. the method generating the solution), the post-hoc Control mutation is significantly less robust than the post-hoc SM-G mutations (Mann-Whitney U-test; $p < 0.001$). The conclusion is that SM-G methods do indeed produce safer mutations in this domain.

6 Reinforcement Learning (RL) Experiments

To explore the scalability of the proposed family of safe mutation operators, this section applies safe mutation techniques to RL domains of increasing difficulty, culminating in first-person traversal of a rendered 3D environment from raw pixels.

6.1 Breadcrumb Hard Maze

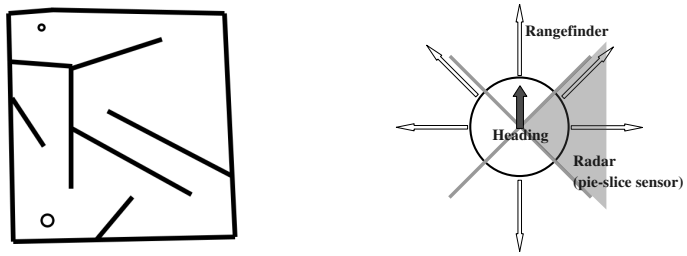
The purpose of this experiment is to explore whether SM approaches have promise for aiding the evolution of deep networks in an RL context. The Breadcrumb Hard Maze (shown in figure 4) was chosen as a representative low-dimensional continuous control task, and is derived from the Hard Maze benchmark of Lehman and Stanley (2011a). An evolved NN controls a wheeled robot embedded within a maze (figure 4a), with the objective of navigating to its terminus. The robot receives egocentric sensor information (figure 4b) 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 in Risi and Stanley (2011) for similar reasons.

Past work applied the NEAT algorithm to this domain, evolving small and relatively shallow NNs (Lehman and Stanley, 2011a; Risi and Stanley, 2011). In contrast, to explore scaling to deep NNs where mutation is likely to become brittle, the NN applied here is deep, consisting of 16 feed forward hidden layers of 8 units each, for a total of 1,266 evolvable parameters. The activation function across hidden layers is the SELU (Klambauer et al., 2017), while the output layer has unsigned sigmoid units. The specific hyperparameters for the NE algorithm and mutation operators are listed in the appendix.

6.1.1 Results

Figure 5 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 was not significantly different 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, supporting the idea that nuanced gradient information may often provided greater benefit to crafting safe variation; similarly, SM-G-SUM performs no differently from the control, suggesting also that the gradient washout effect may extend beyond the toy domain through which it was introduced. The video playlist accompanying this paper contains a video highlighting the qualitative difference between mutations of solutions across mutation operators[1].



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

Figure 4: **The Breadcrumb Hard Maze domain.** The layout of the hard maze is shown in (a), while the maze navigating robot and its sensors is shown in (b). In (a), the large circle represents the starting position of the robot and the small circle represents the goal. Navigating the maze within the 400-step time limit requires efficient behavior. Each arrow outside of the robot's body in (b) is a rangefinder sensor that indicates the distance to the closest obstacle in that direction. The robot has four pie-slice sensors that act as a compass towards the goal, 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.

6.1.2 Large-scale NN Experiment

Follow-up experiments in the Breadcrumb Hard Maze domain are conducted to explore the ability of SM-G methods to evolve large NNs with up to a hundred layers or a million parameters. Note that before now neuroevolution of NNs with over 10 layers has little precedent. Three network architectures inspired by wide residual networks (Zagoruyko and Komodakis, 2016) were attempted, 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 the limits of 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 to explore scaling NE to extreme levels of depth, and each layer is composed of 48 units, for a total of approximately 200,000 parameters. While the previous experiment definitively demonstrates that this level of capacity (a million parameters 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 those same hyperparameters are applied to the 64 and 101-layer models.

Figure 6 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.

[1] `https://goo.gl/BrKDTj`

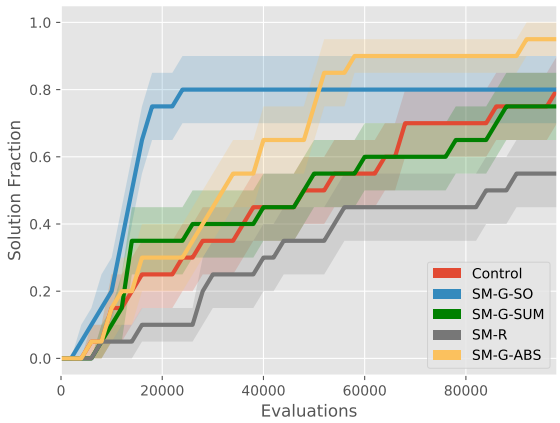
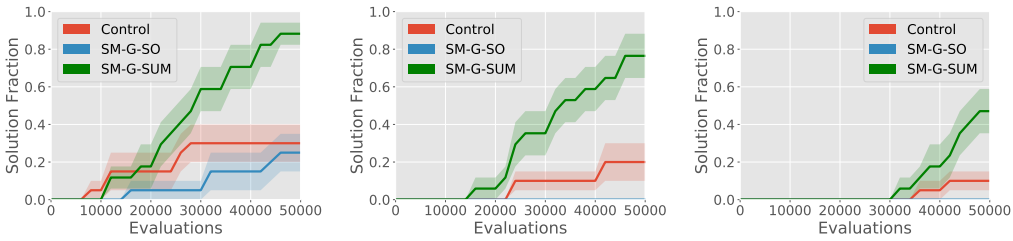


Figure 5: **Performance on the Breadcrumb Hard Maze across methods.** The fraction of solutions evolved by each method is shown over increasing evaluations. SM-G-SO evolves solutions significantly more quickly than the standard mutation control, and only SM-G-ABS evolves solutions in each of its 20 independent 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).



(a) 32 layers (b) 64 layers (c) 101 layers

Figure 6: **Performance on the large-scale NN task across methods.** The fraction of solutions evolved by each method over 20 runs is shown over increasing evaluations. 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 (a) 32-layer, (b) 64-layer, and (c) 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.

6.2 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 (Hausknecht and Stone, 2015; Pathak et al., 2017; Mirowski et al., 2016).

In this domain, the agent receives a visual 64x64 input containing a local view of the 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 7) has many identical intersections, which the agent cannot distinguish by local visual information alone (figure 8), 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.

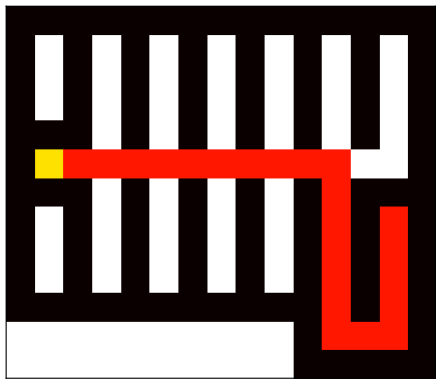
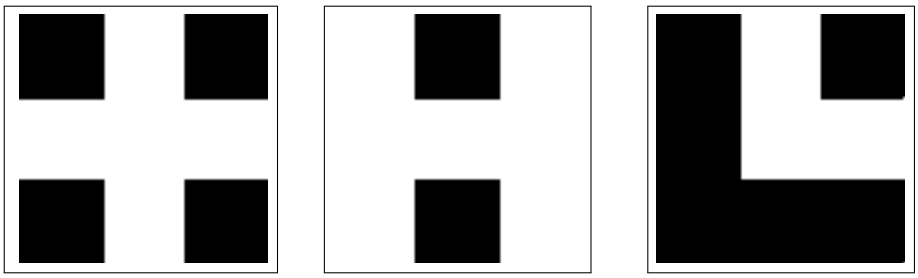


Figure 7: **Topdown Maze domain.** The 2D grid-world maze is shown in which the agent is embedded. A black square indicates a wall and the red path indicates the target trajectory of the agent. The agent receives one fitness point for each square along the trajectory it touches. Note that the red trajectory is not visible to the agent.



(a) View in intersection (b) View between intersections (c) View near end of maze

Figure 8: **Representative agent input in Topdown Maze domain.** The agent views only a 3×3 block window around its immediate location as it proceeds through the maze (figure 7). As a result, many (a) intersections and (b) 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.

6.2.1 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 for this domain 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. All convolutional layers have 12 feature maps. This convolutional core feeds into an average pooling layer that transitions into a two-layer LSTM (Hochreiter and Schmidhuber, 1997) recurrent network with 20 units each, which connects to 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 appendix.

6.2.2 Results

Figure 9 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 5.2, the conclusion from this experiment is that the tested SM-G approaches can accelerate evolution of successful memory-informed navigation behaviors.

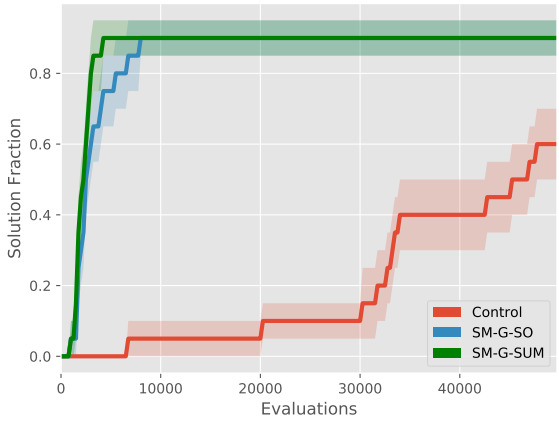


Figure 9: **Performance on the Topdown Maze across methods.** The fraction of successful independent runs (from the 20 conducted for each method) is shown across SM-G methods and the control mutation method. SM-G-SUM and SM-G-SO solve the task consistently and in relatively few evaluations when compared to the control.

6.3 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 10). 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 (Tasfi, 2016).

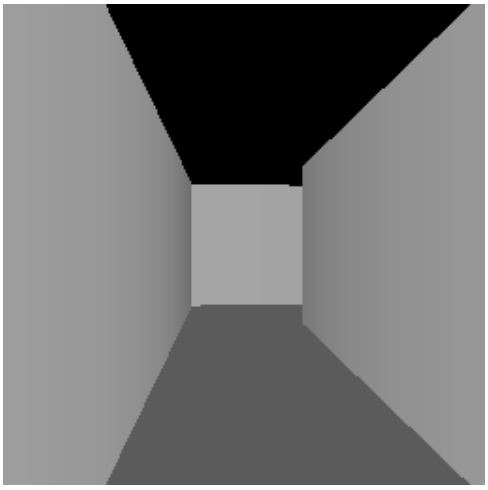


Figure 10: **First-person 3D Maze Domain.** An agent traverses a 3D first-person environment and is rewarded the further it progresses along the correct path.

6.3.1 Experimental Settings

Input to the deep NN is a 64×64 image which is preprocessed only by conversion to grayscale from RGB. The NN for this domain has a similar architecture as in the Topdown Maze, i.e. a deep convolutional core (12 layers of 8 feature maps each) that feeds into a two-layer LSTM stack. 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 has the same settings as in the Topdown maze, although hyperparameters for each mutation method are fit for this domain separately and are described in the appendix.

6.3.2 Results

Figure 11 shows the results in this domain over 30 independent runs. 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 can be viewed from the earlier footnoted video playlist URL.

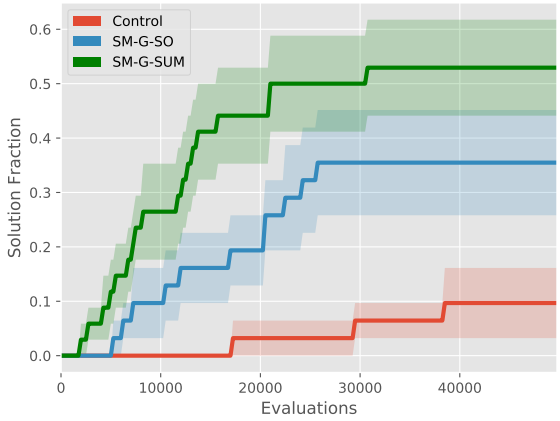


Figure 11: **Performance on the First-person 3D Maze across methods.** The fraction of successful independent runs (from the 30 conducted for each method) is shown across SM-G methods and the control mutation method. SM-G-SUM and SM-G-SO both solve the task significantly more frequently than does the control.

7 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 (unlike most methods in deep learning) the proposed methods do not take performance into account, they can easily be applied to NE-based artificial life or open-ended evolution domains, where a consistent reward signal may not even exist; safe mutation techniques also make no assumptions about the stationarity or uni-modality of reward, and can thus be easily applied to divergent search techniques (Lehman and Stanley, 2011a; Mouret and Clune, 2015; Pugh et al., 2016) and coevolution (Popovici et al., 2012). Additionally, SM-G may productively interact with differentiable *indirect encodings* such as CPPNs (Stanley, 2007), where mutations can have outsize impact; for example, in HyperNEAT (Stanley et al., 2009b; Gauci and Stanley, 2010), CPPN mutations could be 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. (2017) that an evolution strategy (ES) can rival more conventional deep reinforcement learning (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. (2017) itself has an inherent drive towards a form of safe mutations (Lehman et al., 2017), 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 (Stanley and Miikkulainen, 2002), 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 (Liu et al., 2017; Miikkulainen et al., 2017). 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 (beyond only ES), thereby opening the field anew to exploring novel algorithms and ideas.

In principle the increasing availability of parallel computation should present a boon to 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 (Glorot et al., 2011) (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 (Salimans et al., 2017; Petroski Such et al., 2017), 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 (2017)) 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, it is likely that 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 create more productive evolutionary variation. Highlighting another interesting future research direction, preliminary experiments explored a version of SM-G that exploited supervised learning to program an 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, it is important to note that in this paper's experiments, networks of the depth evolved with SM-G have never been evolved before with NE, and those with similar amounts of parameters have rarely been evolved (Petroski Such et al., 2017); 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 all of the members of Uber AI Labs, in particular Thomas Miconi and Xingwen Zhang for helpful discussions. We would also like to thank Justin Pinkul, Mike Deats, Cody Yancey, and the entire OpusStack Team inside Uber for providing resources and technical support.

References

Bedau, M. A., McCaskill, J. S., Packard, N. H., Rasmussen, S., Adami, C., Green, D. G., Ikegami, T., Kaneko, K., and Ray, T. S. (2000). Open problems in artificial life. *Artificial Life*, 6:363–376.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521:503–507.

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*.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1:47–62.

Gangwani, T. and Peng, J. (2017). Genetic policy optimization.

Gauci, J. and Stanley, K. O. (2010). Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 22(7):1860–1898.

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.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gruau, F. (1994). *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France.

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.

Hausknecht, M., Lehman, J., Miikkulainen, R., and Stone, P. (2014). A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hornby, G. S. and Pollack, J. B. (2001). Body-brain co-evolution using l-systems as a generative encoding. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 868–875. Morgan Kaufmann Publishers Inc.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*.

Koutník, J., Schmidhuber, J., and Gomez, F. (2014). Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 541–548. ACM.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Langton, C. G. (1989). Artificial life.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lehman, J., Chen, J., Clune, J., and Stanley, K. O. (2017). ES is more than just a traditional finite-difference approximator. *arXiv preprint to appear*.

Lehman, J. and Miikkulainen, R. (2013). Neuroevolution. *Scholarpedia*, 8(6):30977.

Lehman, J. and Stanley, K. O. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

Lehman, J. and Stanley, K. O. (2011b). Improving evolvability through novelty search and self-adaptation. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 2693–2700. IEEE.

Lehman, J. and Stanley, K. O. (2013). Evolvability is inevitable: Increasing evolvability without the pressure to adapt. *PLoS ONE*, 8(4):e62186.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

Meyer-Nieberg, S. and Beyer, H.-G. (2007). Self-adaptation in evolutionary algorithms. *Parameter setting in evolutionary algorithms*, pages 47–75.

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*.

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*.

Mouret, J. and Clune, J. (2015). Illuminating search spaces by mapping elites. *ArXiv e-prints*, abs/1504.04909.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MIT Press, Cambridge.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*.

Pelikan, M., Goldberg, D. E., and Lobo, F. G. (2002). A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21(1):5–20.

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 to appear*.

Popovici, E., Bucci, A., Wiegand, R. P., and De Jong, E. D. (2012). *Coevolutionary Principles*, pages 987–1033. Springer Berlin Heidelberg, Berlin, Heidelberg.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. 3(40).

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE.

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.

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.

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.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2):131–162.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009a). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009b). A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127.

Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130.

Tasfi, N. (2016). Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

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.

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.

Wilke, C. O., nad Charles Ofria, J. L. W., Lenski, R. E., and Adami, C. (2001). Evolution of digital organisms at high mutation rates leads to survival of the flattest. *Nature*, 412:331–333.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Domain	Control	SM-G-SUM	SM-G-ABS	SM-G-SO	SM-R
Simple Model	0.01	0.5	0.5	0.5	0.5
Recurrent Parity	0.05	0.001	0.001	0.001	0.005
Breadcrumb Hard Maze	0.05	0.1	0.005	0.01	0.005
Large-scale NN	0.01	0.1		0.01	
Topdown Maze	0.1	0.01		0.01	
First-person Maze	0.005	0.01		0.01	

Table 1: **Mutation intensity settings for each domain and method combination.**

Domain	Population Size	Tournament Size	Maximum Evaluations
Simple Model	1 (hill-climber)	NA	2k
Recurrent Parity	250	5	100k
Breadcrumb Hard Maze	250	5	100k
Large-scale NN	100	5	50k
Topdown Maze	250	5	50k
First-person Maze	250	5	50k

Table 2: **Evolutionary hyperparameters across domains.**

Appendix A Hyperparameters

For each experimental domain (besides the simple didactic model in section 5.1), hyperparameter tuning was performed independently for each method. In particular, each method has a single hyperparameter corresponding to mutational intensity. For the control and the SM-G methods, this intensity factor is the standard deviation of the Gaussian noise applied to the parent's parameter vector (which the SM-G methods then reshape based on sensitivity, but which the control leaves unchanged). In contrast, for SM-R, mutational intensity is given by the desired amount of divergence.

Hyperparameter search was instantiated as a simple grid search spanning several orders of magnitude. In particular, each mutational method was evaluated for 8 independent runs with each hyperparameter setting from the following set: $\{1e-1, 5e-2, 1e-2, 5e-3, 1e-3, 1e-4\}$. The best performing hyperparameter was then chosen based on highest average performance from the initial runs, and a final larger set of independent runs was conducted to generate the final results for each domain and method combination. These final hyperparameter settings are shown in table 1. Other hyperparameters (such as population size and tournament size) were fixed between methods and largely fixed between domains, and were subject to little exploration. These hyperparameters are shown in table 2.