Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks

In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016)*. New York, NY: ACM
Nominated for Best Paper Award in Evolutionary Machine Learning.

Gregory Morse
Department of Computer Science
University of Central Florida
Orlando, FL 32816
gregorymorse07@gmail.com

Kenneth O. Stanley
Department of Computer Science
University of Central Florida
Orlando, FL 32816
kstanley@cs.ucf.edu

ABSTRACT

While evolutionary algorithms (EAs) have long offered an alternative approach to optimization, in recent years backpropagation through stochastic gradient descent (SGD) has come to dominate the fields of neural network optimization and deep learning. One hypothesis for the absence of EAs in deep learning is that modern neural networks have become so high dimensional that evolution with its inexact gradient cannot match the exact gradient calculations of backpropagation. Furthermore, the evaluation of a single individual in evolution on the big data sets now prevalent in deep learning would present a prohibitive obstacle towards efficient optimization. This paper challenges these views, suggesting that EAs can be made to run significantly faster than previously thought by evaluating individuals only on a small number of training examples per generation. Surprisingly, using this approach with only a simple EA (called the *limited evaluation EA* or LEEA) is competitive with the performance of the state-of-the-art SGD variant RMSProp on several benchmarks with neural networks with over 1,000 weights. More investigation is warranted, but these initial results suggest the possibility that EAs could be the first viable training alternative for deep learning outside of SGD, thereby opening up deep learning to all the tools of evolutionary computation.

CCS Concepts

•**Computing methodologies** → **Neural networks; Genetic algorithms;** Artificial intelligence; Supervised learning by regression;

Keywords

neural networks; deep learning; machine learning; artificial intelligence; pattern recognition and classification

GECCO '16, July 20–24, 2016, Denver, Colorado, USA

DOI: http://dx.doi.org/10.1145/2908812.2908916

1. INTRODUCTION

Artificial neural networks (ANNs) have witnessed a renaissance in recent years within the field of machine learning through the rise of *deep learning* [6, 22, 27, 33]. The main ideas behind this new approach encompass a range of algorithms [5, 22], but a key unifying principle is that an ANN with multiple hidden layers (which make it deep) can encode increasingly complex features in its upper layers. Interestingly, ANNs were historically trained through a simple algorithm called *backpropagation* [40], which in effect applies *stochastic gradient descent* (SGD) or one of its variants to the weights of the ANN to reduce its overall error, but until about 2006 it was widely believed that backpropagation would lose its gradient in a deep network. Yet discoveries in the last few years have proven that in fact with sufficient training data and processing power backpropagation and SGD turn out to be surprisingly effective at optimizing massive ANNs with millions or more connections and many layers [8, 21, 27]. This realization has in turn led to substantive records being broken in many areas of machine learning through the application of backpropagation in deep learning [8, 21, 27], including unsupervised feature learning [4].

The success of a principle as simple as SGD in achieving record-breaking performance is perhaps surprising. After all, in a space of many dimensions, SGD should be susceptible to local optima, and unlike an evolutionary algorithm, all its eggs are essentially in a single basket because it works in effect with a population of one. Yet it turns out empirically that SGD is penetrating farther towards optimality in networks of thousands or millions of weights than any other approach. Attempting in part to explain this phenomenon, Dauphin et al. [11] make the intriguing argument that in fact very high-dimensional ANN weight spaces provide so many possible escape routes from any given point that local optima are actually highly unlikely. Instead, they suggest that the real stumbling blocks for SGD are *saddle points*, or areas of long, gradual error plateaus. This insight has both helped to explain why SGD might not get stuck, and also to improve it to move faster along such saddle points in the search space. There are also variants of SGD such as RMSProp [48] that help to extricate it from similar situations.

While such analyses may help to explain the success of SGD, they also raise an important question for evolutionary computation (EC): If there are indeed so many paths towards relative optimality in a high-dimensional ANN weight space, then why would not the very same benefits received from this scenario by SGD also apply to EC? In fact, maybe evolutionary algorithms (EAs) should even have an ad-

vantage. After all, a population is perhaps better suited than a single individual to a situation with many promising branches, and simple EAs are agnostic about the rate of descent with respect to the slope of the gradient, which could in principle avoid the kind of saddle-point problem contemplated by Dauphin et al. [11]. In short, the arguments for why SGD can succeed in extremely high-dimensional spaces seem on the face of it to support or even favor EAs as well.

However, because the population in an EA is in effect an approximation of the gradient, it may seem that EAs could be significantly disadvantaged by the fact that they do not compute *exact* gradients, which is precisely what SGD does. However, results have been reported to suggest that the exactitude of the gradient in SGD is not the crux of its success. For example, recalling the application of mutation in EAs, Lillicrap et al. [31] report the surprising discovery that the error signal in a deep network can be multiplied by *random synaptic weights* (entirely breaking the precision of the gradient computation) with little detriment to learning. This result suggests that in fact there are so many viable paths in the high-dimensional space that exactitude is *not* the key causal explanation for reaching near-optimality. Furthermore, given that any search space can be deceptive [18, 49], it is likely often the case that the steepest descent at any given point is not on the shortest path to the optimum anyway. Perhaps it would be even better to maintain a population of options to avoid any premature committal to the best-looking path of the moment.

In fact, the smooth application of SGD in deep learning remains a work in progress. Many tricks have been developed to improve its performance, such as the introduction of rectified linear units (ReLUs) for activation functions, which improves the passing of the gradient from layer to layer over sigmoidal units [38]. Yet even then, researchers continue to observe challenges with finding the right parameters to make such structures learn smoothly, leading to complicated work-arounds like interpolating between different architectures over the course of learning [1] and the recent *highway networks* architecture of Srivastava et al. [43] that in effect turns some neurons on and off over the course of learning, which is reminiscent of evolutionary algorithms that learn structure like NEAT [44]. Thus there remains ample room for new approaches, yet few have considered that such new approaches might come from *outside* SGD.

Perhaps one reason simple EAs have not yet been applied widely to optimizing the weights of deep networks is that most problems in deep learning encompass a large number of training examples. Just as an example, the MNIST image classification database [28] includes 60,000 training examples. While SGD can cycle through these examples on its single learner and adjust its weights based on every individual example, in an EA every individual in the population at every generation must seemingly be evaluated on all the examples to assess its fitness on the training set. Thus a single generation of e.g. 100 individuals would process *six million* examples only to facilitate a single step of the search algorithm.

However, the algorithm introduced in this paper, called the *limited evaluation evolutionary algorithm* (LEEA), is based on a novel insight into the analogy between EAs and SGD that implies that in fact just as an iteration of SGD does not necessitate passing through the entire training set, *neither does a generation of an EA*. Instead, consider that if SGD can compute an error gradient from a single instance (or a small batch of them) then a generation of evolution can be tasked with doing precisely the same. That is, a generation of the EA can be considered analogous to a single iteration of SGD, aiming merely to adjust the weights of the best current approximation(s) to improve with respect to a single instance or small set of them. In this view, the population of 100 might only need to process 100 instances in one generation (instead of six million), which with simple parallelization could in principle be done in the same time it takes to process a single example (and no backpropagation of error need be computed either). Thus the EA begins to look computationally comparable to SGD.

Experiments in this paper on high-dimensional optimization of ANNs will indeed reveal the surprising conclusion that a simple EA appears about as effective as backpropagation through state-of-the-art SGD in problems of over 1,000 dimensions. The competitive performance of the EA in these problems suggests that further research in higher-dimensional neural network optimization is warranted because of the potential for an alternative training strategy in deep learning. This possibility is not just about leveling the playing field with SGD. Rather, it is exciting because EAs bring with them an entirely new toolbox that suddenly becomes applicable to the field of deep learning. Whereas in deep learning researchers apply tricks like regularization for sparsity [16] or dropout [42], EAs have distinctly different options completely unavailable to SGD such as architecture evolution like in NEAT [44], diversity maintenance techniques like novelty search [29], or indirect encodings for ANNs like in HyperNEAT [15, 47]. Thus the entrance of EAs as an alternative to SGD in deep learning would carry with it a broad set of new possibilities.

2. BACKGROUND

The application of EAs to optimizing neural networks is often called *neuroevolution* by its practitioners [12, 44]. As documented in reviews such as by Yao [53] and later Floreano et al. [12], the field of neuroevolution originated in the 1980s (e.g. Montana and Davis [34]), at a time when backpropagation was on the rise [40]. Interest in neuroevolution really picked up in the 1990s, during which a wide variety of approaches were introduced [53]. In these early years, many researchers focused on the intriguing idea (now for the most part long abandoned) that evolution might in fact exceed the capabilities of backpropagation.

In fact, a number of early studies showcase neuroevolution through a variety of EAs matching [51] or even outperforming backpropagation in classification problems [17, 34, 39]. In fact, in these early years enthusiasm was high in part because the future potential of evolving topology along with connection weights seemed to some a significant possible advantage for neuroevolution. As Mühlenbein [37] put it, "We conjecture that the next generation of neural networks will be genetic neural networks which evolve their structure." Many researchers echoed this enthusiasm [7, 10]. Others touted the potential for combining topology evolution with backpropagation [3, 50].

However, as computational capabilities increased over the ensuing decade, researchers began to recognize that the apparent advantages of neuroevolution on relatively simple, low-dimensional problems (i.e. requiring relatively small networks) with small amounts of data might be eclipsed as the

amount of data and size the neural networks increases. For example, even before deep learning really began to showcase the power of SGD with big data, Mandischer [32] begins to articulate the more negative outlook (focusing on Evolution Strategies, or ESs, which are a kind of EA): "We will see that ESs can only compete with gradient-based search in the case of small problems and that ESs are good for training neural networks with a non-differentiable activation function." (It is important to note of course that researchers at the time had not tried the idea in the present paper of only evaluating on a very small number of instances per generation.)

As SGD and backpropagation increasingly dominated the world of classification, especially after the advent of deep learning [5, 22], a significant shift in attention away from classification took hold in the neuroevolution literature. Researchers began to focus on types of problems where backpropagation is more challenging to apply, such as reinforcement learning problems requiring recurrent connections and specialized architectures [2, 13, 20, 35]. As the focus in neuroevolution largely shifted towards reinforcement learning and away from classification, a new generation of neuroevolution algorithms such as NEAT [44, 46], HyperNEAT [15, 47], and a modified CMA-ES [25] gained popularity in part by focusing on the daunting challenge of finding the right architecture for complicated control and decision-making problems, for which SGD provides little answer. Thus the aspiration of EAs to compete directly with SGD in training neural networks for state-of-the-art classification and supervised learning has gradually become only a memory.

Nevertheless, the classic results from the 1990s where neuroevolution does outperform SGD on simple supervised problems [17, 34, 39] remain an intriguing prelude to the ensuing decades of SGD dominance in supervised learning. Despite the seeming clarity of SGD's subsequent dominance in deep learning, the question of why the early promising results of neuroevolution so dramatically fizzled out is actually not well understood. It may seem that neuroevolution is simply definitively not suited to high-dimensional optimization (even with statistical approaches such as in CMA-ES [25] or EDAs [26], which still do not compute exact error gradients), but the support for such a hypothesis is largely speculative because we do not fully understand how or why the structure of neural network search spaces is necessarily vastly more favorable to SGD, which faces its own perils with local optima and saddle points [11]. In short, why should an approximation of the gradient (provided by an EA's population) be any less useful than the single exact gradient computation of a single individual in SGD, which is subject to deception? Both must be imperfect, but given that SGD still succeeds despite the danger of deception, the high-dimensional landscape of neural optimization appears to offer a forgiving landscape of many viable paths, which might be similarly favorable to evolution. This paper therefore revives the old hope from the 1990s that even simple neuroevolution can compete with SGD.

3. APPROACH

A key property of SGD is that the gradient does not have to be calculated for the network over the entire training set. Instead, the gradient can be calculated for a single or very few training examples at a time, which greatly reduces computational cost compared to training on all training examples at once and reduces the chance of becoming stuck in local optima. Interestingly, this ability to adjust weights after very few examples is not necessarily exclusive to gradient descent. Rather, it can in principle apply to any algorithm that traditionally evaluates its solutions against an entire training set, such as EAs.

LEEA implements this idea for the first time in a very simple traditional generational EA. Instead of evaluating the population against the entire training set each generation, the population is evaluated against only a limited number of training examples each generation. This lean approach to evaluation greatly relieves the computational load, particularly on large training sets.

However, one potential weakness of LEEA is that performance on a single example may not correspond to performance across all examples. In SGD, this problem of deceptive examples is tempered by the learning rate and the fact that the population of in effect one individual is never "replaced" by a defective mutant, which prevents the network from shifting too far towards a globally poor configuration during any given evaluation. In contrast, an EA does not inherently contain such safeguards against such deceptive training examples. For example, in the EA, a species well suited to only the present example could displace one that had mastered all the examples before. Two strategies in LEEA mitigate this problem. The first is simple: the population is evaluated against more than one example per generation, though still very few. This strategy reduces the potential damage caused to the population by a single deceptive training example.

It should be noted that while the technique of evaluating the population against a small set of examples each generation may appear to be analogous to a technique employed by SGD called "mini-batching" [9], its motivation in LEEA is different. SGD employs mini-batches primarily to better utilize parallel computational resources, while LEEA employs these mini-batches for more stable population dynamics.

The way the examples are selected for each mini-batch naturally can influence the effectiveness of the algorithm. If all of the examples selected for a given mini-batch happen to have a similar expected output, then even degenerate networks that output a constant value may achieve a high fitness. Instead, if the examples are selected so that they have a diversity of expected outputs, then networks will only be highly rewarded when they can also produce a diversity of outputs that match the expected values. This approach thereby prevents degenerate networks from ever achieving a high fitness and rewards networks that exhibit heterogeneous behavior.

Even with carefully selected mini-batches, LEEA might still lose individuals who are relatively fit in a global sense, but weak on the examples encountered during any single mini-batch. This danger is particularly acute during early evolution, when the best individuals may only succeed on a small percentage of all examples. To further combat this complication, the second key strategy in LEEA is that fitness is calculated based on both the performance on the current mini-batch and the performance of the individual's *ancestors* against their mini-batches. As long as each step of evolution is not changing the behavior of each network in a radical way, this *fitness inheritance* builds up for those individuals who are more likely to be more globally fit than their peers, regardless of how they performed on the current mini-batch.

It is important to note that this form of fitness inheritance is inspired by though differs from previous approaches to fitness inheritance that aimed to avoid directly evaluating a portion of the population [14, 41]. To implement fitness inheritance in LEEA, the fitness for individuals produced by sexual reproduction and asexual reproduction, respectively, is given by

$$f' = \frac{f_{p_1} + f_{p_2}}{2}(1 - d) + f, \text{and} \quad (1)$$

$$f' = f_{p_1}(1 - d) + f, \quad (2)$$

where f' is the individual's modified fitness, f is the fitness of the individual against the current mini-batch, f_{p_n} is the fitness of parent n, and d is a constant decay value.

The introduction of output-diverse mini-batching and fitness inheritance enables LEEA to take advantage of the computational benefits of SGD within the framework of evolutionary computation. Other than these two strategies, LEEA is just a simple generational EA with a mutation power decay (which is analogous to learning rate decay in SGD):

Algorithm 1 LEEA

1: **while** $gen < maxGenerations$ **do**
2: select mini-batch of training examples
3: evaluate population
4: modify fitnesses based on fitness inheritance
5: select parent(s) from a truncated list with roulette wheel selection
6: create offspring – sexual reproduction with uniform crossover (without mutation) or asexual reproduction with uniformly distributed mutation
7: reduce maximum mutation power by multiplying by decay constant
8: $gen = gen + 1$
9: **end while**

This algorithm contains only the bare essentials required for a more traditional EA to work, along with the modifications necessary to combat the complications caused by limited evaluations per generation.

4. EXPERIMENT

To assess the effectiveness of LEEA as an alternative to more traditional neural network optimization methods, performance is tested in three domains against a traditional generational EA (TGEA), SGD, and RMSProp, which is a cutting-edge variant of gradient descent based on the idea that following shallow gradients can often be as useful as following steep ones [48]. The TGEA works like LEEA, but with *all* training examples evaluated in each generation (instead of a mini-batch), and no fitness inheritance mechanism. In RMSProp, the enhancement to SGD, the current gradient information is divided by a running average of recent gradients. This technique allows the algorithm to escape from plateaus with tiny gradients more quickly. All three domains are chosen because they can benefit from a neural network of moderate dimensionality, i.e. over 1,000 connection weights that must be optimized. In general it is not a common view that a simple EA is suited to optimizing so many parameters in a neural network as effectively as SGD. Therefore these experiments demand an unusual ability for EAs that is rarely contemplated in deep learning literature.

Sample data from each domain is divided into a training set, a validation set, and a test set. All algorithms evaluate performance against the validation set to test for overfitting. The evolution-based algorithms additionally require this validation performance data to determine which individual from the population is selected for evaluation against the test set. All reported results are thus based on performance against the test set and for evolution the individual tested is the one that performed best against the validation set.

The first domain, with 800 training examples, is a function approximation task, where the function is given by the equation

$$f(x, y) = \frac{sin(5x(3y + 1)) + 1}{2}. \quad (3)$$

Preliminary tests indicate that the surface generated by this function is sufficiently difficult to approximate that networks with over 1,000 connections have an advantage over smaller networks, thus making a good test for the effective use of high dimensions.

The second domain is a time series prediction task with 1,200 training examples where the time series is generated using the Mackey-Glass chaotic time series equation

$$\frac{dx}{dt} = \beta \frac{x_r}{1 + {x_r}^n} - \gamma x, \quad \gamma, \beta, n > 0 \quad (4)$$

where $\gamma = 1$, $\beta = 2$, $r = 2$, $n = 9.65$, $x_0 = 1.1$, and $x_1 = 1.2$. The prediction for time t is based on values at $t - 6$, $t - 12$, $t-18$, and $t-24$. This task has been employed to test neural networks in the past [24], and is also sufficiently complex to potentially benefit from over 1,000 connections.

The third domain is the California housing dataset, which is a housing value prediction task with 10,000 training examples also previously given to neural networks [24, 30]. This task contains observations of housing data where each observation consists of eight input attributes for a particular block of housing (median income, median house age, etc.) and one output for the median house value.

All learning algorithms train with the same network topology, which contains two hidden layers with 50 nodes in the first layer and 20 nodes in the second layer, as well as a single output. Because of differences in the number of inputs, the resultant networks have 1,170, 1,270, and 1,470 connections for the first, second, and third domains, respectively (networks include also a bias node). LEEA and TGEA are both evolved using a direct encoding (i.e. one floating-point gene per connection in the network) with no ability to change the network topology.

To evaluate the effectiveness of each algorithm, they are exposed to the same total number of examples during training. That way, the number of generations given to TGEA equals the number of epochs given to SGD and RMSProp, while LEEA is given a proportionately higher number of generations (which of course still means the same number of evaluations) than TGEA based on the ratio of total training examples to the number of examples evaluated per generation. Because the EAs must evaluate all members of their

	Function Approximation	Time Series	California Housing
TGEA	0.1643 ± 0.0121	0.2068 ± 0.0107	0.1216 ± 0.0013
LEEA	0.0711 ± 0.0116	$\mathbf{0.1080 \pm 0.0340}$	0.1147 ± 0.0020
SGD	$\mathbf{0.0539 \pm 0.0087}$	0.1336 ± 0.0380	0.1097 ± 0.0016
RMSProp	0.0636 ± 0.0138	0.1227 ± 0.0410	$\mathbf{0.1092 \pm 0.0007}$

Table 1: RMSE with standard deviation for each algorithm on each domain averaged over 10 runs. (Lower is better.)

population, on a single processor their execution time becomes greater proportionally to the population size. However, parallelization can potentially benefit EAs to a greater degree because each such evaluation is independent. As the capacity for parallelization increases, the instrumental issue thus shifts from population size to the ability to effectively learn from each training example, which is why performance is measured here based on the number of examples evaluated.

Parameters for all algorithms were selected through a parameter sweep on the first domain. There were two key differences found between the ideal parameters of LEEA compared to TGEA. For LEEA, the ideal starting mutation power (the maximum size of a weight mutation) is 0.03, compared to 0.1 for TGEA. This difference reflects that it is better to take smaller steps when evaluating individuals on a small set of training examples. In addition, while TGEA performs best with fitness sharing-based speciation [19] in the spirit of NEAT [44] based on genetic similarity (which maintains diversity), LEEA performs best without any speciation. This observation makes sense because the changing mini-batch in each generation is a force for diversity on its own.

Parameters in common between LEEA and TGEA are population size (1,000), mutation power decay (0.99), mutation rate (0.04), selection proportion (0.4), and sexual reproduction proportion (0.5). Selection proportion refers to the ratio of individuals (sorted by fitness) that are eligible for reproduction.

Ideal mini-batch size in LEEA was determined experimentally to be 2. In all three domains, there is no notion of output classes, but rather a single real number output in the range $\{0, 1\}$. To apply the idea of diversity in mini-batches, the output range is divided into two equally sized sections, and examples selected so that each mini-batch contains one example for both sections of the range. A new mini-batch is randomly generated at the beginning of each generation (while ensuring the diversity of the expected outputs). The ideal fitness inheritance decay rate for LEEA was determined experimentally to be $d = 0.2$.

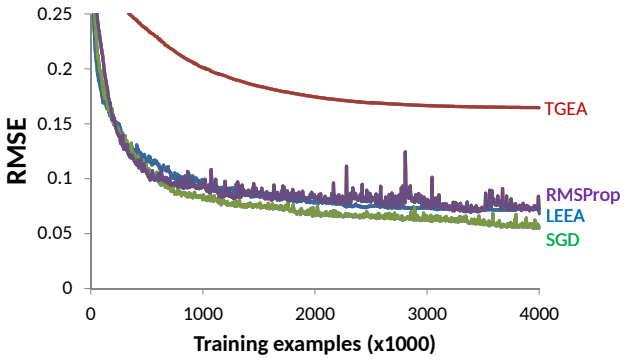
SGD and RMSProp were implemented as online learning (i.e. one example at a time). Mini-batching with these methods is not included in the comparison because there is not a consensus on an empirical advantage for mini-batching on a per-epoch basis, and some have even suggested perhaps a slight disadvantage [52]. Therefore, the online case serves as a more transparent baseline that avoids introducing any confounding factors that might come with mini-batching. The training examples are reshuffled at the start of each epoch. The initial learning rate for SGD and RMSProp is 0.3 and 0.001, respectively. The learning rate is decayed exponentially by 0.0001 per epoch for both SGD and RMSProp. RMSProp maintains a queue of 250 recent magnitudes for each parameter for calculating the per-parameter learning rate.

All neurons in every setup are sigmoidal based on the function $f(x) = \frac{1}{1+e^{-x}}$. That way, all methods are compared in equivalent conditions. Of course it is known that other functions like rectified linear units [38] can sometimes help deep networks under SGD, but the aim here is to establish how these methods behave under equivalent controlled conditions to get a sense of their relative capabilities in general. Source code for the experiments in this paper can be found at http://eplex.cs.ucf.edu/uncategorised/software.

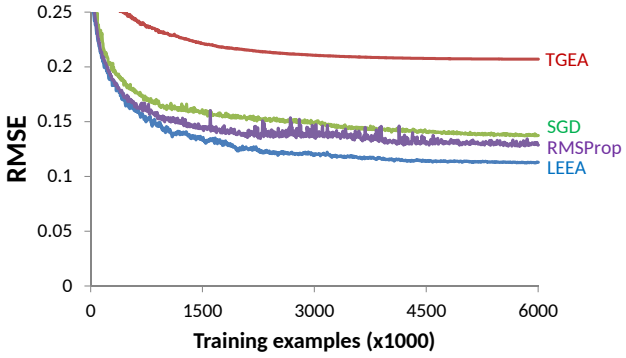
5. RESULTS

Table 1 shows testing results for each algorithm on each domain. Overall test performance in the table is measured as the root mean square error (RMSE) of the individual who performs best against the validation set across the run, where individuals are tested against the validation set at regular intervals based on the number of training examples in the domain (every 4,000 samples for function approximation, every 6,000 samples for the time series, and every 30,000 samples for California Housing). The individual tested in the evolutionary runs is the one who performs best on validation from the population at the current generation. It is important to note that overall, each algorithm experiences precisely the same number of examples before each validation check, so the amount of training data experienced is always equivalent. Test performance is averaged over 10 runs for each algorithm/domain combination. Some results did not exhibit a normal distribution, so significance between results is calculated with the Mann-Whitney U test. On the function approximation domain, LEEA's performance is not statistically significantly different from RMSProp, but it performs slightly though significantly worse than SGD ($p < 0.01$) and significantly better than TGEA ($p < 0.01$). On the time series domain, though LEEA performs best by a small margin, there is no statistical difference between LEEA, SGD, and RMSProp, and all three algorithms perform significantly better than TGEA ($p < 0.01$). On the California housing domain, LEEA performs slightly though significantly worse than SGD and RMSProp ($p < 0.01$) and significantly better than TGEA ($p < 0.01$). As a whole, comparing results between LEEA and TGEA in all the domains confirms that EA performance can be significantly enhanced by evaluating a limited number of examples per generation. They also show that differences between LEEA, SGD, and RMSProp range from very small to insignificant, with LEEA even performing best (though insignificantly) in one domain, supporting the conclusion that LEEA is a viable option for training neural networks.

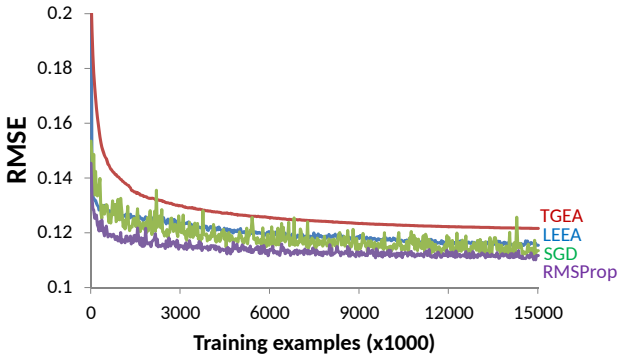
To provide further insight into these results, figure 1 shows average testing accuracy across whole runs, with the x-axis normalized such that all algorithms are evaluated against an



(a) Function approximation



(b) Time series



(c) California housing

Figure 1: Average root mean square error on the test set over time (lower is better). The average performance over total training examples seen so far as measured against the test set is displayed for each algorithm on the (a) function approximation, (b) time series, and (c) California housing domains. The main result is that the performance of LEEA closely matches that of SGD and RMSProp.

equal number of training examples on the same problem at the same point along x. For the evolution-based algorithms, the whole population is evaluated against the validation set at each measurement interval and the individual with the best validation performance is selected for testing. While TGEA learns more slowly and converges to a less optimal solution than SGD and RMSProp, LEEA exhibits a qualitatively similar training curve to these conventional deep learning training algorithms. This observation provides further evidence that LEEA offers an evolution-based alternative for training complex neural networks that is potentially competitive with the state of the art.

6. DISCUSSION AND CONCLUSION

The results suggest that a simple EA with limited evaluations in each generation (the LEEA) can optimize neural networks of over 1,000 dimensions about as effectively and in about the same number of iterations as gradient descent algorithms that currently dominate the field of deep learning. While these results do not prove that such EAs will continue to work well on the neural networks of millions or more weights that now feature in the most cutting-edge results in deep learning [21], they are an intriguing hint that the potential for EAs in this area may be greater than previously believed. At the least it suggests that investigation of such algorithms on larger networks is warranted.

The core question on much larger networks is whether somehow the gradient over the high number of dimensions becomes too hard to approximate for the EA. For example, if such high-dimensional optimization requires effectively sampling changes along nearly every dimension, the EA population might fail to sample densely enough. Yet if it is true that there are many paths to near-optimality as has been argued even in deep learning literature [11], it may not ultimately be essential to sample even a large proportion of the possible paths. In fact, sparse sampling could be a winning trade-off as the price to gain diversity, which conventional SGD lacks. In any case, only further empirical investigation on more complex domains such as MNIST [28] can settle this question.

Interestingly, even if sampling is ultimately too sparse, there is always the option of expanding the population size. Continuing advances in hardware and the availability of graphics processing units (GPUs) foreshadow the possibility of parallelizing much larger populations in the future. While GPUs have recently been celebrated for their parallelism largely in the context of SGD within deep learning, EAs where individuals can be evaluated separately are particularly suited to large-scale parallelization. In the future, GPUs and EAs could present a particularly powerful marriage. Furthermore, performance might be improved in the future also by refining the method for computing fitness inheritance.

The potential for EAs to offer an alternative to SGD for training neural networks is compelling because EAs are a genuinely different paradigm for specifying a search problem. That is, the real payoff of such a novel option is not that it might perform better in some scenario, but that it allows researchers to approach problems in entirely different ways and capitalize on different forms of regularization, thereby greatly expanding the toolbox available to neural network researchers.

For example, expressing the loss function for SGD to target precisely the desired behavior can be harder than for a fitness function. Consider a two-class classification problem where a factory product is either *working* or *defective*. While it is relatively straightforward to configure SGD to minimize error on the classification, it would be much harder to minimize *monetary cost* for making different kinds of mistakes: misclassifying a defective product as working might be a lot more expensive than misclassifying a working product as defective. The advantage of the fitness function is that

it can *directly* minimize even an indirect cost because it is intrinsically more expressive.

Furthermore, EAs offer options unavailable to SGD like diversity maintenance [36], the evolution of topology [44], and indirect encoding [15, 45]. These can all act as powerful regularizers different from those in SGD. Problems facing SGD in backpropagation like vanishing gradients through multiple layers or through recurrence also would not even exist for neuroevolution through EAs, perhaps enabling radically different architectures to be learned than the ones designed for SGD like LSTMs [23], or even autoencoders [4].

The possibility of such a dramatically different training paradigm is intriguing, and the empirical evidence in this paper offers a hint that it is at least sufficiently conceivable to warrant further serious investigation. The properties of search spaces with millions of dimensions are still not fully understood, leaving open the chance that something quite different than backpropagation can play a constructive role in such spaces as well.

Acknowledgments

This work was supported by the National Science Foundation under grant no. IIS-1421925. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Special thanks to the UCF Stokes cluster for providing computation resources.

References

[1] D. Almeida and N. Sauder. GradNets: Dynamic interpolation between neural architectures. *ArXiv e-prints*, abs/1511.06827, 2015.

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1994.

[3] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *Artificial Life II: Proceedings of the Workshop on Artificial Life*, 1990.

[4] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1): 1–127, 2009.

[5] Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. *Large-Scale Kernel Machines*, 34, 2007.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS)*, Cambridge, MA, 2007. MIT Press.

[7] H. Braun and J. Weisbrod. Evolving neural feedforward networks. In *Artificial Neural Nets and Genetic Algorithms*, pages 25–32. Springer, 1993.

[8] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[9] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *Advances in neural information processing systems*, pages 1647–1655, 2011.

[10] D. Dasgupta and D. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96, 1992.

[11] Y. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *ArXiv e-prints*, abs/1406.2572, 2014.

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

[13] D. B. Fogel. *Blondie24: Playing at the Edge of AI*. 2001.

[14] L. G. Fonseca, A. C. Lemonge, and H. J. Barbosa. A study on fitness inheritance for enhanced efficiency in real-coded genetic algorithms. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2012.

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

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

[17] C. Goerick and T. Rodemann. Evolution strategies: An alternative to gradient-based learning. In *Proceedings of the International Conference on Engineering Applications of Neural Networks*, volume 1, pages 479–482. Citeseer, 1996.

[18] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In L. Davis, editor, *Genetic algorithms and simulated annealing*, pages 74–88, London, 1987. Pitman.

[19] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[20] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5: 317–342, 1997.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[22] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

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

[24] G.-B. Huang, P. Saratchandran, and N. Sundararajan. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. *Neural Networks, IEEE Transactions on*, 16(1):57–67, 2005.

[25] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 2588–2595, Piscataway, NJ, 2003. IEEE Press.

[26] P. Larranaga and J. A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2002.

[27] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning (ICML-2012)*, 2012.

[28] Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 1998.

[29] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

[30] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *Neural Networks, IEEE Transactions on*, 17 (6):1411–1423, 2006.

[31] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[32] M. Mandischer. A comparison of evolution strategies and backpropagation for neural network training. *Neurocomputing*, 42(1):87–117, 2002.

[33] R. Marc'Aurelio, L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1185–1192, Cambridge, MA, 2007. MIT Press.

[34] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. pages 762–767, 1989.

[35] D. E. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive co-evolution. *Evolutionary Computation*, 5:373–399, 1997.

[36] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

[37] H. Mühlenbein. Limitations of multi-layer perceptron networks – steps towards genetic neural networks. *Parallel Computing*, 14(3):249–260, 1990.

[38] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[39] V. W. Porto, D. B. Fogel, and L. J. Fogel. Alternative neural network training methods. *IEEE Intelligent Systems*, (3):16–22, 1995.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, pages 318–362. 1986.

[41] R. E. Smith, B. A. Dike, and S. A. Stegmann. Fitness inheritance in genetic algorithms. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, SAC '95, 1995.

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[43] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *ArXiv e-prints*, abs/1505.00387, 2015.

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

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

[46] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research (JAIR)*, 21: 63–100, 2004.

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

[48] T. Tieleman and G. Hinton. Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[49] D. Whitley. Fundamental principles of deception. *Foundations of Genetic Algorithms (FOGA 1)*, 1:221, 1991.

[50] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14: 347–361, 1990.

[51] W. Wienholt. Minimizing the system error in feedforward neural networks with evolution strategy. In *ICANN 93*, pages 490–493. Springer, 1993.

[52] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

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