Genetically-Trained Deep Neural Networks

Krzysztof Pawełczyk
Silesian University of Technology
Gliwice, Poland
krzysztof.pawelczyk@polsl.pl

Michal Kawulok
Silesian University of Technology
Gliwice, Poland
michal.kawulok@polsl.pl

Jakub Nalepa
Silesian University of Technology
Gliwice, Poland
jakub.nalepa@polsl.pl

ABSTRACT

Deep learning is a widely explored research area, as it established the state of the art in many fields. However, the effectiveness of deep neural networks (DNNs) is affected by several factors related with their training. The commonly used gradient-based back-propagation algorithm suffers from a number of shortcomings, such as slow convergence, difficulties with escaping local minima of the search space, and vanishing/exploding gradients. In this work, we propose a genetic algorithm assisted by gradient learning to improve the DNN training process. Our method is applicable to any DNN architecture or dataset, and the reported experiments confirm that the evolved DNN models consistently outperform those trained using a classical method within the same time budget.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Genetic algorithms**;

KEYWORDS

Genetic algorithm, deep learning, convolutional neural network

ACM Reference Format:
Krzysztof Pawełczyk, Michal Kawulok, and Jakub Nalepa. 2018. Genetically-Trained Deep Neural Networks. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3205651.3208763

1 INTRODUCTION

Various methods for training deep neural networks (DNNs) were developed over the years. In particular, gradient-based approaches were explored, despite their serious disadvantages such as slow convergence, high time complexity, difficulties with escaping local minima of the search space, and vanishing/exploding gradients. They were addressed with numerous enhancements, including new activation functions [1], advanced weights initialization [6], stochastic optimization techniques, and transfer learning [7].

Recently, evolutionary algorithms have also been explored to improve the DNNs, namely (i) to train the weights [4] (hence substituting the gradient-based learning), (ii) to evolve their architectures [3, 5], or (iii) to optimize their hyper-parameters [2].

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

ACM ISBN 978-1-4503-5764-7/18/07.
https://doi.org/10.1145/3205651.3208763

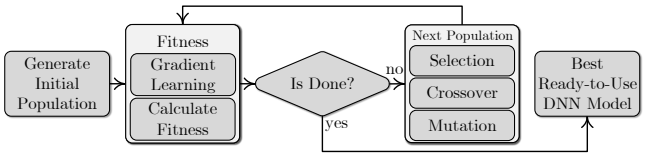

Figure 1: Flowchart of the proposed method.

In this paper, we focus on addressing the main downsides of the gradient-based approaches. Our contribution lies in combining their strengths with the benefits of evolving the weights, which has not been reported in the literature so far. We alternate a genetic algorithm (GA) with short sessions of gradient-based training to elaborate the DNN weights. The weights of the parents are inherited by their offspring, so short gradient-based learning performed to evaluate each individual is sufficient to fine-tune the model. Our approach helps avoid local minima in the search space and improves the convergence, hence allows for fast and effective DNN training. This is confirmed by the experimental results reported in this paper.

2 PROPOSED METHOD

We propose a new DNN learning scheme, which exploits a GA in cooperation with short sessions of the back-propagation algorithm (Figure 1). The number of gradient learning iterations ϕ is an integral part of an individual, and it is adaptively evolved.

The **initial population** is composed of $\mathcal{N}$ individuals (each of which encodes a DNN) with random weights. The initial value of ϕ is drawn from $\langle \xi_{min}, \xi_{max} \rangle$ with uniform distribution. To compute the **fitness** of each individual, the associated DNN, initialized by its parents weights, is fine-tuned for ϕ iterations, and the fitness is retrieved as the classification error δ_{train} over the training set.

In **selection**, the individuals are ranked according to δ_{train}, top $\mathcal{E}$ constitute an elite which always survives to the next population. The rest is randomly paired with the elite individuals, yet those with $\delta_{train} > \omega$ are replaced by random solutions from the elite.

Parental solutions are recombined using a **single-point crossover** of each DNN layer, performed independently from each other as shown in Figure 2. For the convolutional layers, the crossover point is chosen between the convolutional kernels. Weight matrices of the fully-connected (FC) layers are crossed over row-wise. The

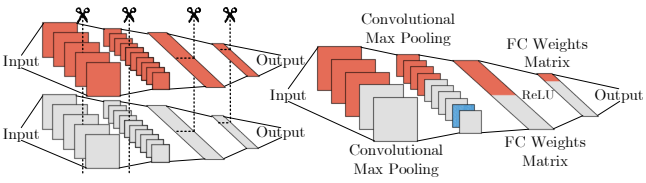

Figure 2: Example of crossover between two DNN topologies (LeNet-4). The kernel rendered in blue is mutated.

Table 1: Values of the parameters used during the experiments in all scenarios.

The GA parameters were tuned experimentally to the following values: $\mathcal{N} = 10$, $\mathcal{E} = 3$, $\omega = 80\%$, the crossover and mutation rates were set to 0.8 and 0.2, respectively.

	Adaptive repeated 30×			Scenarios with fixed ϕ, all of the individuals initialized with the same value, repeated 10×														
	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}^{5}_{const}$	$\mathcal{S}^{25}_{const}$	$\mathcal{S}^{100}_{const}$	$\mathcal{S}^{200}_{const}$	$\mathcal{S}^{400}_{const}$	$\mathcal{S}^{600}_{const}$	$\mathcal{S}^{800}_{const}$	$\mathcal{S}^{1000}_{const}$	$\mathcal{S}^{1200}_{const}$	$\mathcal{S}^{1400}_{const}$	$\mathcal{S}^{1600}_{const}$	$\mathcal{S}^{1800}_{const}$	$\mathcal{S}^{1875}_{const}$	$\mathcal{S}^{2000}_{const}$
$\mathcal{F}$	1.0	1.2	2.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ξ_{min}	1	230	102	5	25	100	200	400	600	800	1000	1200	1400	1600	1800	1875	2000	
ξ_{max}	1875	280	152	5	25	100	200	400	600	800	1000	1200	1400	1600	1800	1875	2000	

Table 2: The number of iterations (ϕ), convergence time, δ_{train} and δ_{test} for adaptive scenarios. The best scores are bold.

	ϕ	Time (s)	δ_{train} (%)	δ_{test} (%)
$\mathcal{S}_1$	1627 ± 219	814±121	**.00 ± .00**	.80 ± .06
$\mathcal{S}_2$	457 ± 353	1339±314	**.00 ± .00**	**.78 ± .05**
$\mathcal{S}_3$	945 ± 698	1403±650	**.00 ± .00**	.78 ± .06

number of training iterations (ϕ) is encoded within each individual as a binary string, and is subject to a single-point crossover as well.

Mutation modifies only one part of an individual at a time. Mutation in a convolutional layer consists in re-initializing one randomly-chosen kernel. An FC layer mutates by randomizing a single row of its weights matrix. The ϕ is mutated by single bit flip operation, or by multiplication by a factor $\mathcal{F}$.

3 EXPERIMENTS

The algorithms were implemented in C++ with the Caffe2 framework, and validated on a computer with an Intel Core i7 7800X with 64GB RAM and NVIDIA GeForce GTX 1080Ti. We used the stochastic gradient descent, with the base learning rate set to .1, updated with each iteration by a factor of .999. Table 1 presents the GA setups investigated during our experiments. Each experiment was time-framed for one hour. Also, if at least one individual converges to $\delta_{train} = .0\%$, then the evolution is terminated.

To validate our method, we used a well-established MNIST set which contains grayscale images of hand written digits (0-9). The dataset is divided into two balanced and non-overlapping subsets: training $\mathcal{M}_{train}$ with $6 \cdot 10^4$ samples, and test $\mathcal{M}_{test}$ of 10^4 samples. The vanilla LeNet-4 DNN was used—although our method is agnostic with respect to the underlying architecture (and can be easily applied to any DNN), we focused on a fairly straight-forward and simple network [2]. DNNs were trained using mini-batches (32 examples), hence the number of gradient-based learning iterations needed to process all of the training samples is $m_{train} = 1875$.

Within the assumed time budget, DNN achieved $\delta_{train} = .95\%$ and $\delta_{test} = 1.15\%$ using the back-propagation algorithm (we treat this as a baseline). Table 2 shows that in adaptive scenarios, our algorithm converges with $\delta_{train} = .0\%$ and δ_{test} is lower than for the gradient-based learning, within half of the given time budget. In the $\mathcal{S}_1$ variant, the GA chooses individuals with high ϕ. Analysis of the intermediate populations allows us to conclude that DNNs with $\phi > 1400$ become the majority of the population in 4.31 ± 2.15 generations, because they easily outperform the rivals. All $\mathcal{S}_1$ processes reached $\delta_{train} = .0\%$ in a fairly similar convergence time.

The results for $\mathcal{S}_2$ and $\mathcal{S}_3$ show that our GA keeps the population more diversified in terms of ϕ (the differences across the ϕ values are statistically important for all pairs of investigated GA variants at $p < .01$, Wilcoxon test), which allows the GA to explore a larger part of the solution space than $\mathcal{S}_1$. However, it converges slower, and the required time is also more varied. Although there is no significant difference in the generalization performance of DNNs evolved by our adaptive GAs ($p > .1$), the best model was retrieved using $\mathcal{S}_3$ ($\delta_{test} = .59\%$—almost half of the baseline error rate).

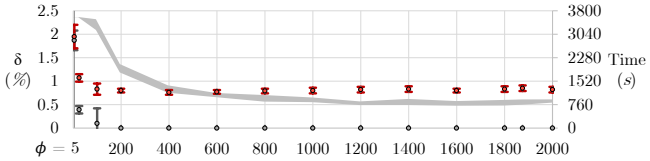

Figure 3: The range of time values (gray area), δ_{train} (gray dots) and δ_{test} (red dots) errors for different values of ϕ.

In Figure 3, we report the results for $\mathcal{S}^{\phi}_{const}$ with a fixed number of ϕ. Even for extremely short gradient-based learning sessions ($\phi < 100$), our GA finds a DNN which is able to achieve decent levels of δ_{train} and δ_{test}, however it may not outperform the baseline in a given time. The $\mathcal{S}^{400}_{const}$ obtained the lowest δ_{test}. The processing times per each $\mathcal{S}^{\phi}_{const}$ decrease with larger ϕ, as the algorithm sooner finds the DNNs with $\delta_{train} = .0\%$. The fastest in providing the results was $\mathcal{S}^{1600}_{const}$, after which the trend is reversed.

The multi-start nature of a GA induces wider exploratory of the search space and reduces the risk of staying in local minima. The crossover operator combines learned layers of parental DNNs, which is similar to the transfer learning known to speed up the convergence [7]. In cooperation with the mutation operator, which randomly reinitializes one section of one network layer, it also prevents feature extractors (kernels) from being co-adapted.

4 CONCLUSIONS

In this paper, we introduced a new DNN learning scheme which couples a GA with the gradient-based algorithm. Our method is highly generic, scalable, and it can be applied to any network architecture. Although it has been validated using only one multi-class benchmark dataset, it is evident that high-quality DNN models can be evolved. In all experimental scenarios, our technique outperformed classical gradient-based back-propagation learning procedure.

ACKNOWLEDGMENTS

This work was supported by the National Science Centre under Grant DEC-2017/25/B/ST6/00474, and by the Silesian University of Technology, Poland, funds no. BKM-509/RAu2/2017.

REFERENCES

[1] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. 2013. Maxout Networks. In *Proc. ICML '13*, Vol. 28. PMLR, 1319–1327.
[2] P. Ribalta Lorenzo, J. Nalepa, M. Kawulok, L. Sanchez Ramos, and J. Ranilla Pastor. 2017. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proc. GECCO '17*. ACM, 481–488.
[3] R. Miikkulainen and et al. 2017. Evolving Deep Neural Networks. (2017). arXiv:1703.00548v2
[4] F. Petroski Such, V. Madhavan, E. Conti, and J. Lehman. 2017. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. (2017). arXiv:1712.06567v2
[5] M. Suganuma, S. Shirakawa, and T. Nagao. 2017. A Genetic Programming Approach to Designing CNN Architectures. In *Proc. GECCO '17*. ACM, 497–504.
[6] W. Sun, F. Su, and L. Wang. 2018. Improving DNN with multi-layer maxout networks and a novel initialization method. *Neurocomputing* 278 (2018), 34 – 40.
[7] J. Wang. 2018. Bimodal Vein Data Mining via Cross-Selected-Domain Knowledge Transfer. *IEEE Trans. on Information Forensics and Security* 13 (2018), 733–744.