Comparison of Neural Network Optimizers for Relative Ranking Retention Between Neural Architectures

George Kyriakides, Konstantinos Margaritis

Comparison of Neural Network Optimizers for Relative Ranking Retention Between Neural Architectures

George Kyriakides[1][0000-0002-5877-0943] and Konstantinos Margaritis[0000-0002-1750-5115]

University of Macedonia, Thessaloniki 55236, Greece
[1]ge.kyriakides@uom.edu.gr

Abstract. Autonomous design and optimization of neural networks is gaining increasingly more attention from the research community. The main barrier is the computational resources required to conduct experimental and production project. Although most researchers focus on new design methodologies, the main computational cost remains the evaluation of candidate architectures. In this paper we investigate the feasibility of using reduced epoch training, by measuring the rank correlation coefficients between sets of optimizers, given a fixed number of training epochs. We discover ranking correlations of more than 0.75 and up to 0.964 between Adam with 50 training epochs, stochastic gradient descent with nesterov momentum with 10 training epochs and Adam with 20 training epochs. Moreover, we show the ability of genetic algorithms to find high-quality solutions of a function, by searching in a perturbed search space, given that certain correlation criteria are met.

Keywords: Deep Learning, Neural Architecture Search, Ranking.

1 Introduction

Optimization and autonomous design of neural architectures have been increasingly investigated in the past years [1-8]. Partly due to the popularity of deep learning [9-11] and partly due to the cumbersome process of designing and optimizing the networks. It is only natural that we wish to free ourselves of the repetitively unique process of designing and optimizing. Many methods have been proposed.

Although most studies propose methods that produce better networks or converge faster, most of them are concerned with the way that the design or optimization algorithm acts. Nonetheless, the majority of computational resources are consumed during the candidate architectures' evaluation. Thus, many researchers distribute the workload among many machines in order to speed up the process. Although a possible and quite effective solution, not all algorithms favor distributed setups. Furthermore, it requires availability of additional hardware.

On the other hand, the networks are evaluated only to further the search or optimization method, until the final architecture is produced. As such, the actual performance of the network does not matter as an absolute value. It matters in relation to the other alternatives that the design algorithm has at each point in time. As such, methods that

enable a quick relative evaluation of network architectures can greatly reduce the computational workload of designing and optimizing architectures.

In this paper, we study the use of various neural network optimizers under a small number of training epochs as a proxy of relative network performance. Furthermore, we conduct Monte-Carlo simulations of searching for the best parameters in 3-d spaces with similar correlation coefficients. The motivation behind this work is to examine the feasibility of using faster evaluation schemes (reduced number of epochs) when evaluating architectures in an autonomous design framework, where absolute performance has a smaller impact than relative performance between candidate architectures.

2 Related Work

As stated previously, most research focuses on faster and more efficient methodologies. Nonetheless, much of this work has greatly contributed to the field. A well-known approach entails the use of genetic algorithms in order to evolve networks of fixed sub-module design [1]. Two versions are proposed, DeepNEAT which is an extension of the older NEAT [12] method for fully-connected networks and CoDeepNEAT which utilizes the co-evolution [13] of graph structure and sub-module populations.

Another approach, utilizes reinforcement learning in order to design CNN architectures [2] and transferable architectures [3]. In [4] an attempt to reduce the number of trainable parameters by freezing all network layers minus the output layer, results in a positive relative ranking. This indicates that freezing intermediate layers is a viable approach for neural architecture search. Another study attempts to design architectures directly on the target dataset, without the use of proxy tasks [7]. Although the authors argue that a proxyless search produces better results, it also demands greater computational resources. It is only logical that a search conducted on the target hardware and training conditions will produce the best results. Furthermore, in [1] researchers noticed that by restricting the available training epochs during evolution, the final produced architecture required less epochs to achieve performance comparable to state of the art.

Other approaches employ random search [6], sequential model-based optimization [5] and boosting [8].

3 Methodology

3.1 Evolving Architectures

In this paper we use the methodology proposed in [1], called DeepNEAT to evolve neural architectures. Although the original method uses speciated crossover to produce offspring from a population as well as mutations, we use only the mutation operator, as our goal is to produce an array of different architectures of increasing complexity.

The methodology initializes an individual architecture comprised only of an input and output node. At each mutation, there is a certain probability that a new connection will be added or an existing connection will be deactivated and replaced by a new module. Each module consists of a convolutional layer, a dropout layer, a weight scaling

factor and a max pooling layer with probability of 0.5. Each gene in the chromosome contains information about the module's parameters: the number of filters in the convolutional layer, in the space [32, 256], the dropout rate in [0, 0.7], the weight scaling factor in [0, 2.0], the kernel size in {1, 3} and if the max pooling layer is present or not. In our experiment the probability of adding a new module node was set at 0.05 and the probability of adding a new connection at 0.1. Some of the architectures generated are depicted in **Fig. 1**. Number 1 is the starting architecture, consisting only of an input and an output layer. Each node denotes a whole module. Due to the fact that many node modules must be scaled in order for the network to be functional, Number 20 has 15 modules that translate to 86 PyTorch layers.

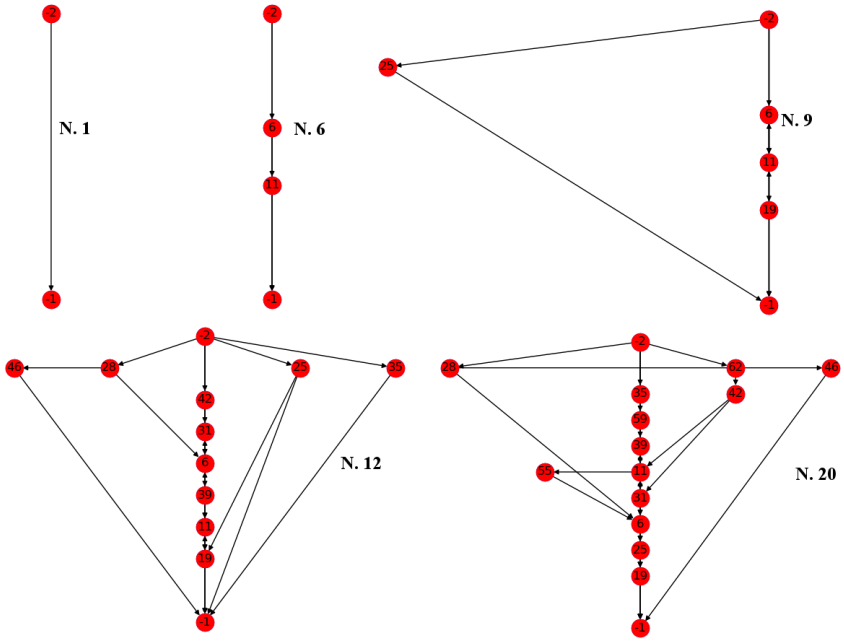

Fig. 1. Sample architectures evolved.

3.2 Experimental Setup

For our experiments, we generated several distinct neural networks of increasing complexity by mutating a simple network using DeepNeat for 20 generations, with the goal of classifying the CIFAR10 image recognition dataset. We trained each network using 1, 5, 10, 20, and 50 epochs respectively. We compared a total of seven optimizers; Adadelta, Adam, Adamax, RMSprop, Stochastic Gradient Descent (SGD), SGD with momentum (SGD-M), and SGD with nesterov momentum (SGD-NM). Following, we calculated each network's rank for its specific optimizer and epochs combination. Our

goal was to compare the relative rankings of low training cost combinations to the relative rankings of fully trained networks (50 epochs).
In order to compare the networks' rankings, we use Kendall's rank correlation coefficient (τ) [14], which is a measure of similarity between ranks (correlation of ranks). High positive or negative values imply correlation (positive and negative respectively). To further investigate the ability of optimizers to discern high-performing architectures, we perform the analysis on all architectures as a whole, as well as the last ten generated architectures. The motivation behind this decision is that simple architectures will exhibit similar behavior and minor differences in performance can be attributed to other factors, such as initial conditions. Complex architectures on the other hand, demand more computational resources to train. Thus, design methods will benefit more from a faster but relatively consistent evaluation.

Following, we conducted Monte-Carlo simulations of 3-d Rastrigin functions with similar correlation coefficients to those observed in our experiments in order to demonstrate the feasibility of using reduced training epochs as proxies for neural architecture search. We employed a simple genetic algorithm in order to find the local minima, both in the original function as well as the perturbed function. We initialized a population of 10 individuals, with crossover rate of 0.9 and mutation rate of 0.02. The population was selected for crossover by tournament selection. We experimented with 10, 100, and 1000 generations. Each experiment was repeated 1000 times, in order to obtain a reasonable distribution sample. Given the multi-modal nature of Rastrigin function, its search space resembles those in neural architectures.

3.3 Implementation

For our experimental implementation we used the NORD benchmarking framework. DeepNEAT was implemented in Python but only the mutation operator was used. Furthermore, we used ESA's Pygmo implementation of a simple genetic algorithm for our Monte-Carlo simulations [15] The experiments were run on 7 Tesla K40 GPUs.

4 Results

4.1 Neural Architectures

In this study our focus is the retention of relative rankings while using reduced training epochs for candidate architectures. Nonetheless, the achieved accuracy of each architecture, optimizer and epoch combination provides us with an intuitive way to assess the quality of each network. **Fig. 2** depicts the accuracy of each optimizer for 1 and 50 epochs of training. It is evident that with 50 epochs the stability is greatly increased. Nevertheless, it seems that there exists a pattern, that both groups follow. A sharp increase in accuracy at generation 10, a relatively stable performance for a small number of generations and then a sharp decrease. This pattern seems to be followed by all optimizers on both groups, except for the 'Ada' family (Adamax, Adadelta, Adam) when given 50 epochs of training. The training time required for each group is depicted in **Table 1**. As it is evident, the number of training epochs seems to be the most influential

factor. Furthermore, required time seems to scale linearly with the number of epochs, as it is expected.

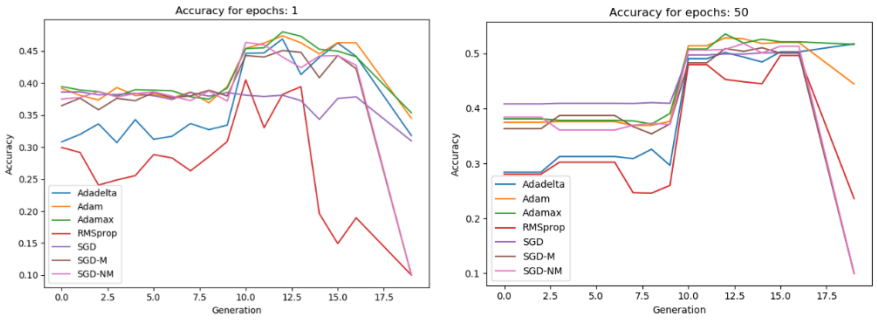

Fig. 2. Comparison of achieved accuracy in 1 epoch of training to 50 epochs.

Table 1. Average training times (in seconds per architecture).

Optimizer	1 Epoch	5 Epochs	10 Epochs	20 Epochs	50 Epochs
Adadelta	36.1	158.6	312.3	622.1	1701.5
Adam	35.9	157.5	311.7	617.4	1724.8
Adamax	35.8	157.1	310.3	617.8	1718.6
RMSprop	35.4	156.5	310.2	614.6	1687.6
SGD	36.0	157.1	308.3	613.2	1687.3
SGD-M	35.7	157.2	309.3	613.5	1709.5
SGD-NM	35.5	156.6	311.2	617.7	1707.4

Although useful, it is not easy to judge relative performance from accuracy plots and training time tables. In order to better compare the relative performance of the various optimizer and training epoch combinations, we ranked the generated architectures, depending on the accuracy that they achieved on the specific epoch-optimizer combination. We then computed the rank correlation coefficient for each list with the list generated by the same optimizer when the architectures are trained for 50 epochs before evaluation. The results show high correlation coefficients (above 0.65) for most combinations, except for RMSprop_01 and SGD_01 as depicted in **Fig. 3** .

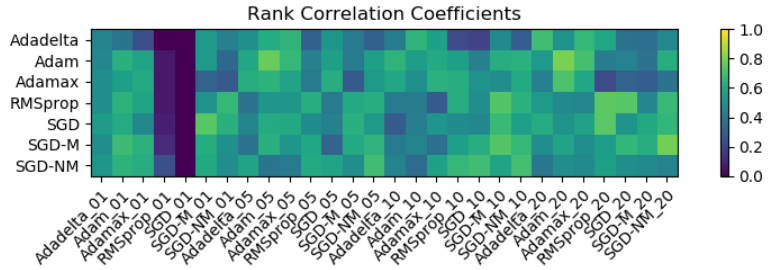

Fig. 3. Rank correlation coefficients heatmap.

The results seem promising, although they do not show a clear or extremely strong correlation between the groups. This can be partly attributed to the fact that simple architectures will perform relatively close, independently of the optimizer and the number of epochs trained. If the number of parameters is small and the problem domain large, there are simply too little degrees of freedom to adequately fit the model. Thus, we repeat the analysis, by discarding the first half of the generated architectures. The goal is to test if rank correlations are retained on more complex architectures. As it is evident in **Fig. 4**, there are now strong relationships between some groups. Adam_01, Adam_05, Adam_20, Adamax_10, Adamax_20, RMSprop_20, SGD-M_10, and SGD-NM_10 have high correlations with Adam ($\tau > 0.75$, $p < 0.05$). Furthermore, SGD-NM_10 and Adam_20 have $\tau = 0.964$ with $p < 0.01$, indicating a strong correlation in relative rankings. This confirms that as architecture complexity increases, evaluation schemes come to a more robust agreement about their relative performance.

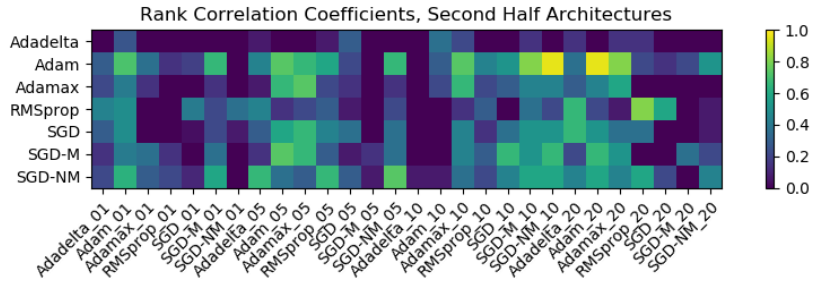

Fig. 4. Rank correlation coefficients heatmap for the second half of generated architectures.

4.2 Monte-Carlo Simulations

We saw the need to further study the ability to optimize through a proxy task with specific rank correlation coefficient, compared to the original search space. We also saw the need to study the behavior of optimization algorithms when they are given more

freedom to explore both search spaces (i.e. they are not restricted by computational resources). In order to do so, a controlled simulation environment had to be created. The environment should have many local optima, similar to the search space of neural architectures. Furthermore, we wanted to search on a proxy space, were the proxy and original spaces had similar correlation coefficient to the one found in our original experiment. This would enable to study the feasibility of using any proxy task that is known to produce correlated results with the original search space. Thus, we generated a 3-d Rastrigin function and added uniform noise, until a fixed rank correlation coefficient was achieved, compared to the original function. We followed the same procedure as in our original experiment. The genetic algorithm searched the space of the perturbed function (proxy task) and the final solution was tested on the original. A 2-d example of the two functions can be seen in **Fig. 5** showing how the addition of noise does change the function's surface, but leaves major features and patterns intact.

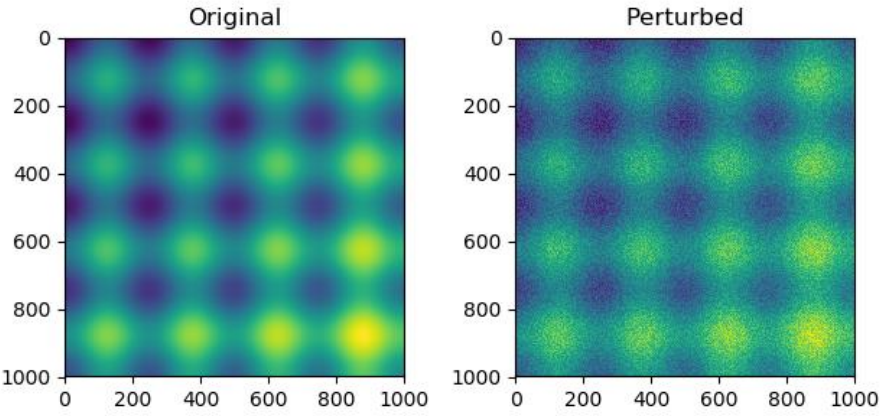

Fig. 5. 2-D Rastrigin functions, original and perturbed with $\tau = 0.738$

Table 2 depicts the average (E) and **Table 3** the standard deviation(σ) of the produced solutions, when applied to the original function. We conducted the experiment for three levels of correlation (τ=0.75, τ=0.85, and τ=0.95). In the original space, the genetic algorithm is able to produce solutions close to the top 1% by searching the space of the unperturbed Rastrigin function. For τ=0.75, by conducting the search on the perturbed spaces and then applying the final solutions on the normal function, the genetic algorithm is able to produce solutions near the top 4%. For τ=0.85 and τ=0.95 it

is able to produce solutions near the top 3% and 2% respectively. By constructing a 95% confidence interval, the original function's space produces a solution that belongs in the top [0, 0.033]. For the worst-case scenario (τ=0.75), the perturbed function's space produces solutions in the top [0.016, 0.08]. For τ=0.85, the solutions lie in [0.006, 0.07] and for τ=0.95 they exist in [0.003, 0.039]. Furthermore, we can see that the solutions' standard deviation reduces as τ increases, given enough generations to search the given space. As it is evident, the most important factor concerning the final solution's quality is the number of generations that the algorithm was allowed to run.

Table 2. Mean of achieved solution's top percentage for the original and perturbed search space, when applied to the original function.

Generations	E(original)	E(τ=0.95)	E(τ=0.85)	E(τ=0.75)
10	0.160	0.162	0.184	0.195
100	0.050	0.058	0.086	0.103
1000	0.013	0.022	0.032	0.040

Table 3. Standard deviation of achieved solution's top percentage for the original and perturbed search space, when applied to the original function.

Generations	σ(original)	σ(τ=0.95)	σ(τ=0.85)	σ(τ=0.75)
10	0.075	0.066	0.075	0.084
100	0.031	0.034	0.045	0.052
1000	0.010	0.012	0.019	0.021

5 Limitations

This study provides some interesting results, both in the real-world experiment, as well as the Monte-Carlo simulation. Nonetheless, more thorough research must be conducted in order to assess the feasibility of using reduced epoch training as a proxy for evaluating relative rankings in neural network architectures. The candidate architectures in this study are admittedly few, even though they are diverse. Moreover, we tested our results in a single image recognition task, although it utilized a popular and well-studied dataset. We hope that in our future work we will overcome some of these limitations, by running longer experiments on more recent hardware, thus being able to employ more architectures and datasets.

6 Conclusions

In this paper we studied the relative ranking correlations of neural architectures by evolving an initial architecture with DeepNEAT's mutation operator and training them

for a set number of epochs, utilizing select optimizers. By comparing the 1, 5, 10, and 20 epoch groups to the 50-epoch group, we observed positive and relatively high correlations for most optimizers, except for the RMSprop_01 and SGD_01 optimizer-epochs combination. In order to test the most complex architectures of our candidates, we split the candidates in two and tested the relative rankings of the second half. This revealed a strong correlation between Adam and the two optimizer-epochs groups, SGD-NM_10 and Adam_20. The correlation coefficient was very high for both groups ($\tau = 0.964$) and showed high statistical significance ($p < 0.01$). In order to test the ability of optimization algorithms to produce optimal solutions from correlated proxy tasks, we employed a genetic algorithm on a perturbed Rastrigin function, with a rank correlation coefficient of 0.75 to the original. The genetic algorithm produced solutions that belong on average, to the top 3,5% of the original function's solutions.

From the above, we conclude that there is a positive probability that reduced epoch training can be used in order to evaluate the relative rankings of neural architectures for design and optimization purposes. It is evident that the design/optimization process does not care for the performance of intermediate solutions, only for their relative performance. Furthermore, genetic algorithms perform relatively well when searching in perturbed spaces, given that the correlation with the original space is at least 0.75. Nonetheless, more thorough research must be conducted in order to establish optimal proxies for neural architecture design and optimization in a variety of domains.

Finally, the most decisive factor concerning the algorithm's solution quality is the number of generations that it is allowed to run. Thus, it can be advantageous to use proxy tasks with lower correlation to the original search space, if the reduction in computational resources required to evaluate a single network can be dedicated to running a more exhaustive search.

7 Future Work

In light of our recent findings, we hope to continue the research into feasible solutions for the relative comparison of neural network architectures. A higher number of candidate architectures and a broader range of datasets will greatly enhance our confidence in the results, given that they agree with the current. Moreover, comparisons with even higher number of training epochs will further enhance our confidence in the results. Finally, a study that employs well-established design methodologies will contribute to the final verdict on the feasibility of using reduced epoch training, by collecting data from the actual problem.

8 Acknowledgements

This work was supported by computational time granted from the Greek Research & Technology Network (GRNET) in the National HPC facility - ARIS - under project ID DNAD. Furthermore, this research is funded by the University of Macedonia Research Committee as part of the "Principal Research 2019" funding program.

References

1. Miikkulainen R, Liang J, Meyerson E et al. (2019) Evolving Deep Neural Networks. Artificial Intelligence in the Age of Neural Networks and Brain Computing 293-312. doi: 10.1016/b978-0-12-815480-9.00015-3
2. Zoph B, Le Q (2016) Neural Architecture Search with Reinforcement Learning. In: arXiv.org. https://arxiv.org/abs/1611.01578. Accessed 21 Feb 2019
3. Zoph B, Vasudevan V, Shlens J, Le Q (2018) Learning Transferable Architectures for Scalable Image Recognition. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. doi: 10.1109/cvpr.2018.00907
4. Kyriakides G, Margaritis K (2018) Neural Architecture Search with Synchronous Advantage Actor-Critic Methods and Partial Training. Proceedings of the 10th Hellenic Conference on Artificial Intelligence - SETN '18. doi: 10.1145/3200947.3208068
5. Liu C, Zoph B, Neumann M et al. (2018) Progressive Neural Architecture Search. Computer Vision – ECCV 2018 19-35. doi: 10.1007/978-3-030-01246-5_2
6. Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. The Journal of Machine Learning Research 13:281-305.
7. Cai H, Zhu L, Han S (2019) ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In: arXiv.org. https://arxiv.org/abs/1812.00332. Accessed 19 Mar 2019
8. Cortes C, Gonzalvo X, Kuznetsov V et al. (2019) AdaNet: Adaptive Structural Learning of Artificial Neural Networks. In: arXiv.org. https://arxiv.org/abs/1607.01097. Accessed 21 Feb 2019
9. Dong C, Shi Y, Tao R (2018) Convolutional Neural Networks for Clothing Image Style Recognition. DEStech Transactions on Computer Science and Engineering. doi: 10.12783/dtcse/cmsms2018/25262
10. Lee D, McNair J (2018) Deep Reinforcement Learning Agent for Playing 2D Shooting Games. International Journal of Control and Automation 11:193-200. doi: 10.14257/ijca.2018.11.3.17
11. Gatys L, Ecker A, Bethge M (2016) Image style transfer using convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
12. Stanley K, Miikkulainen R (2002) Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation 10:99-127. doi: 10.1162/106365602320169811
13. Moriarty D, Miikkulainen R (1997) Forming Neural Networks Through Efficient and Adaptive Coevolution. Evolutionary Computation 5:373-399. doi: 10.1162/evco.1997.5.4.373
14. Kendall M (1938) A New Measure of Rank Correlation. Biometrika 30:81. doi: 10.2307/2332226
15. Biscani F, Izzo D, Jakob W et al. (2019) esa/pagmo2: pagmo 2.10. In: Zenodo. https://doi.org/10.5281/zenodo.1045336. Accessed 21 Feb 2019