Short-and-Long-Term Impact of Initialization Functions in NeuroEvolution

Lucas Gabriel Coimbra Evangelista and Rafael Giusti(✉)

Institute of Computing, Federal University of Amazonas, Manaus, Brazil
{lucas.evangelista,rgiusti}@icomp.ufam.edu.br

Abstract. Neural evolutionary computation has risen as a promising approach to propose neural network architectures without human interference. However, the often high computational cost of these approaches is a serious challenge for their application and research. In this work, we empirically analyse standard practices with Coevolution of Deep NeuroEvolution of Augmenting Topologies (CoDeepNEAT) and the effect that different initialization functions have when experiments are tuned for quick evolving networks on a small number of generations and small populations. We compare networks initialized with the He, Glorot, and Random initializations on different settings of population size, number of generations, training epochs, etc. Our results suggest that properly setting hyperparameters for short training sessions in each generation may be sufficient to produce competitive neural networks. We also observed that the He initialization, when associated with neural evolution, has a tendency to create architectures with multiple residual connections, while the Glorot initializer has the opposite effect.

Keywords: Deep NeuralEvolution · Genetic algorithms · Weight initialization

1 Introduction

Deep Neural Networks (DNNs) are among the most used machine learning methods nowadays. They can be applied in multiple scenarios and are able to approximate functions that are often considered too complex for "classic" models, such as Support Vector Machines and shallow Neural Networks. However, DNNs tend to be complex, so their training usually requires very large datasets and they are computationally expensive. This is particularly challenging for the task of fine-turning hyperparameters, since their validation may take considerable time.

We thank Coordination for the Improvement of Higher Education Personnel - CAPES/PROAP and Amazonas State Research Support Foundation - FAPEAM/POSGRAD 2021. This research was partially supported by CAPES via student support grant #88887.498437/2020-00.

J. C. Xavier-Junior and R. A. Rios (Eds.): BRACIS 2022, LNAI 13653, pp. 298–312, 2022.
https://doi.org/10.1007/978-3-031-21686-2_21

When designing a DNN, several factors must be considered, especially the number and design of the layers. Once the architecture has been chosen, training a DNN for a specific task requires defining several hyperparameters, such as the number of epochs, learning rate, optimization function, batch size, etc. Considering the advances in the last decade in the development of Deep Learning models to deal with challenging tasks and the remarkable effort involved in designing these models "by hand", methods capable of automatically finding ideal DNN architectures without human intervention have been growing increasingly relevant. Many of these advances were possible with the birth of the field of Deep Learning and Bio-Inspired Algorithms, which created a new area of study: Deep Evolutionary Neural Networks, also called Deep NeuroEvolution [1].

Deep NeuroEvolution (DNE) deals with using evolutionary algorithms with the specific purpose of optimizing the architecture of a DNN to solve difficult problems. Like "classical" evolutionary algorithms, the DNE approach employs a population of sub-optimal solutions, called individuals, and the goal is to identify and combine the best elements of the individuals to find solutions that are as close to the optimal as possible. This usually happens in cycles called generations. In the context of DNE, it means combining the most promising elements of neural networks (e.g., layers, neurons, or blocks of layers) at the end of each generation, creating architectures that are increasingly more suitable for some specific task, such as classification, anomaly detection, time series forecasting, and others.

One important issue that has received relatively little attention in NeuroEvolution is the weight initialization. At each generation, the evolutionary algorithm produces neural networks whose weights may need to be optimized from scratch. Different initialization functions may be used to set the initial neuron parameters, but not all of them are equally suitable. A bad initialization choice can lead to some undesirable behaviors such as the vanishing gradient problem and the gradient explosion problem [2,3]. Generally, these phenomena occur when the parameters tend to zero or to infinity, respectively, making it impossible for the machine learning model to converge during the learning process [3,4]. Furthermore, different initialization functions may lead to different architectures.

In the past decade, we have witnessed many alternative optimizations and studies on parameter initializers. These studies draw primarily on two groundbreaking researches that introduced the most commonly used functions at present time: the Glorot initialization [5] and the He initialization [6].

However, these functions were proposed to suit DNNs with a fixed architecture, which is designed "by hand" after meticulous choices on how many layers the network should have and on the layout of those layers. In NeuroEvolution, neural network components are combined in often unexpected ways, and in a sense we can think of the individuals as networks that evolve over time. Furthermore, while in "traditional" application a DNN architecture is defined once and then trained over a reasonable period of time for some specific application, this is not as easily done in NeuroEvolution, where a large number of networks have to be quickly trained at each generation, and the best elements of a DNN must be identified and combined with others to produce better individuals. With these

issues in mind, we ask the following question: how do standard weight initialization functions help to rapidly converge a neural network and find the ideal DNN elements in a scenario where the network architecture is constantly changing? This question is crucial for evolutionary algorithms to become, in fact, competitive to non-evolutionary DNNs, which represent the current state of the art.

The remained of this paper is as follows. Section 2 presents related works that address weight initiatlization or maintenace in DNE. Section 3 explains the theoretical foundation behind this work. Section 4 describes and discusses our results. Finally, Sect. 5 draws final remarks.

2 Related Work

Much of the research effort in DNE is directed towards finding better ways to create ideal topological structures for the target problem, somewhat neglecting the potential benefits of better initializing or updating the weights of the networks during the evolutionary process. In this section, we give attention to some works that tackle the latter issue.

Focusing on *updating* weights, Koutnik et al. [7] created a new method to encode the weights of neural networks using Fourier coefficients. This allows exploring the spatial relationship between the weights and reducing the dimensionality of the target problem. As a main result, Koutnik et al. managed to reduce the total number of iterations to obtain the best individual in three benchmark problems (*pole-balancing*, *ball throwing*, and *octopusarm control*). Togelius et al. [8] decided to focus efforts on the crossover stage during the evolution of architectures, inspired by the workings of memetic algorithms, to find the best combination of weights between individuals rather than a random one. As benchmark they used the Race Car problem, comparing five different algorithms: Hill-Climber, Simultaneous Climber, Memetic Climber, Constrained Memetic Climber, and Inverse Memetic Climber. The authors report that the more features in the input layer (dimensions), the better results the evolutionary versions of the algorithms obtained, always outperforming the non-evolutionary versions. Neither works, however, address the total time to carry out the experiments.

Specifically considering the weight *initialization* problem, Okada et al. [9] proposed to represent the weights of neural networks not as scalars but as intervals, an extension of Evolutionary Strategy for neurevolution of intervalued neural networks, deciding not to evolve the topology of the architectures, and to work only with the prediction of values for sinusoidal functions, considering the intervals of such functions as genotypes. Also in this aspect, Desell [10], applied a new NeuroEvolutionary algorithm, EXACT (*Evolutionary eXploration of Augmenting Convolutional Topologies*), in order to address three possible fronts during the evolutionary process: (i) node-level mutation operations; (ii) epigenetic weight initialization; and (iii) pooling connections. When coining the expression *epigenetic weight initialization*, Desell aimed to investigate a better representation of the combination of the parents' genomes (weights), without necessarily changing the initial combination; in this way, the new individuals have their

weights directly inherited from this epigenetic combination, similarly to [8]. As benchmark, the results were compared on the MNIST database, where the best individual obtained 99.46% accuracy. In all, Desell evolved over 225,000 neural networks with the support of 3,500 volunteers who served as hosts to run the experiments. None of these works comment on the total execution time.

In 2021, Lyu et al. [11] compared the performance of the NeuroEvolutive EXAMM (*Evolutionary eXploration of Augmenting Memory Models*), a gradient-based algorithm for recurrent neural networks, using four distinct functions to initialize the weights of the neural networks: Glorot, He, Uniform Random, and a new method called Lamarckian weight inheritence. To test the performance, EXAMM was applied in time series forecasting on real-world databases. The experiments were performed with 2,304 processing cores (Intel Xeon Gold 6150@2.70 GHz CPU) and a whopping 24 TB of RAM. This is the only work in our review that directly mentions the problem of the exploding and vanishing gradients. However, again the authors do not report the required processing time for their experiments.

3 Theoretical Foundation

3.1 CoDeepNEAT

In this paper we intend to work with the *coevolution of Deep NeuroEvolution of Augmenting Topologies* (CoDeepNEAT) algorithm [12]. As pointed out by Papavasileiou et al. [13], CoDeepNEAT is among some of the most innovative techniques that combine evolutionary algorithms based on non-gradient descent and algorithms based on gradient descent. In CoDeepNEAT, chunks of layers are developed as modules, which are merged together to create a blueprint, which in turn is used to create multiple architectures (Fig. 1).

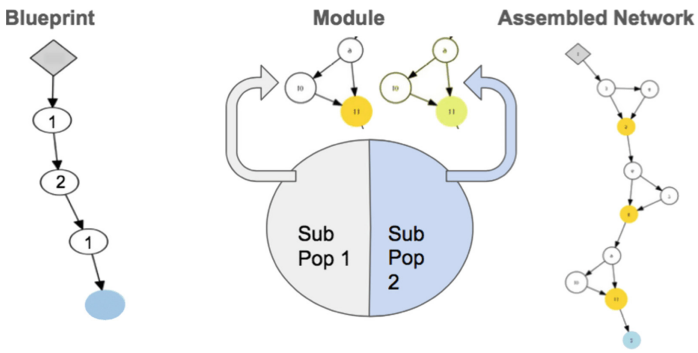

Fig. 1. Assembly of a neural network from a blueprint individual and a set of module individuals [12]. The number inside each node in the blueprint represents species in the modules population. (Color figure online)

The algorithm works with a population of modules and a population of blueprints, which are evolved separately. The blueprint chromosome is a graph where each node contains a pointer to a particular module species, while each module chromosome is a graph that represents a chunk of DNN layers.

Before the first generation, the initial population of modules is generated according to the hyperparameters of the CoDeepNEAT algorithm. A module may contain convolution layers or fully-connected layers, and the probability of getting either one is specified by the user (we keep both at 50%). If the module contains convolution layers, its hyperparameters, such as the number of filters and the kernel size, are chosen randomly from a set of possible values that are also user-specified. In our experiments, the number of filters ranges from $[32, 256]$ and the kernel size is either 1, 3, or 5.

At the beginning of each generation, a number of blueprints are randomly chosen from the blueprint population. Each node in a blueprint is replaced with a module that is randomly chosen from the module population. This results in a set of assembled neural network that comprise the individuals of that generation. Those neural networks are untrained, and their weights must be initialized, even if they come from modules that did not change since the previous generation. Each individual is then trained on a small number of epochs and evaluated on a hold-out partition of the training dataset.

Since CoDeepNEAT works with two populations, there are two fitness values to be calculated. The fitness of each blueprint individual is the average metric (loss, accuracy, F1-score etc.) of all neural networks assembled from that individual. If the blueprint was not chosen to sprout neural networks in this generation, then its fitness value is zero. Similarly, the fitness of each module is the average metric of all networks that employed that module as part of their architecture.

At the end of each generation, except for the last one, new blueprints are generated from the crossover of the best individuals, producing a new population of blueprints. The same is done to the population of modules. After the last generation is completed, the NeuroEvolution algorithm is ready to present to the user the indicated DNN for some particular task. This may be the individual with best fitness value from the final generation, or the best individual from all generations. This individual is then trained with the entire dataset for a suitable number of epochs, which is usually significantly larger than the number of epochs used to calculate the fitness of blueprints and modules during evolution.

The size of the modules population is a CoDeepNEAT hyperparameter. In our experiments, they are either 30 or 45, and the size of the blueprints population is either 10 or 25, as explained in Sect. 4.

3.2 Short, Medium and Long Term Analyses

A major drawback of bioinspired deep learning models is that they are very time consuming. In spite of their very competitive results, the time required for the evolution of architectures can make reproducibility difficult and is a challenge for researchers without access to high computational power. To put in perspective, Bohrer et al. [14] give an estimated 480+ h to reproduce the CIFAR-10 dataset

experiments from the inaugural work of the CoDeepNEAT [12]. They reach that figure from the assumption that each epoch takes at least 30 s, and each of the 100 DNNs in one generation is trained for 8 epochs over a total of 72 generations.

The obvious way to reduce the required time is to reduce the complexity of the neural networks. In [14], the hyperparameter space is drastically reduced. To begin with, they evolved the networks for only 40 generations, instead of the 70 initially employed in [12]. The number of neural networks, blueprints, and modules was also reduced, as well as the complexity of the modules (fewer filters, larger possible kernels, and no max pooling). With a smaller search space, they also reduced the amount of training data. The CIFAR-10 dataset contains 50,000 training instances. While [12] uses all of them on a hold-out scheme to train and validate the DNNs, [14] employs only 40% of that data.

In this work, we consider training the neural networks during merely four generations, and we employ the same population sizes and use as much data for training as [14]. This is what we refer to as "short-term" evolution of neural networks. The main motivation behind this experiment is to verify whether an initialization function presents advantage over others when individuals are evaluated after very short training episodes. We also verify whether ReLU or hyperbolic tangent is more suitable an activation function in the short-term.

Subsequently, we also analyze the behavior of CoDeepNEAT in the medium and long term. In both cases, we use the entirety of the training dataset, with a train/validation hold-out partition, to evolve neural networks. And we increase the resources available to CoDeepNEAT in both instances.

Another argument worth mentioning to justify larger evolutionary hyperparameters is the emergence of skip connections with output summation, which resemble residual architectures, according to Miikkulainen et al. [12]. It is known that the loss function tends to be chaotic in very deep architectures, which is unfavorable for trainability and hinders generalization. Residual architectures are very popular in deep learning [15–17] because they tend to simplify that search space. This was experimentally validated in [18], which shows that the loss landscape changes significantly when skip connections are introduced, as illustrated in Fig. 2.

The residual connections implementation presents an uninterrupted flow of gradient from a given layer to the one closest to the output, avoiding the problem of vanishing gradient. Consequently, multiple skip connections are an alternative to ensure the reuse of resources of the same dimensionality as the previous layers. On the other hand, these connections are also useful for recovering spatial information lost during downsampling, and seem to stabilize gradient updates in very deep architectures, ensuring rapid convergence. Such structure is so important that it was the main motivator used by Miikkulainen et al. to introduce mutation of connections between neurons in the original CoDeepNEAT [12,19].

3.3 Initialization and Activation Functions

This work focuses on the analysis of the results obtained with coevolutionary algorithms when different initialization functions are employed. In spite of its

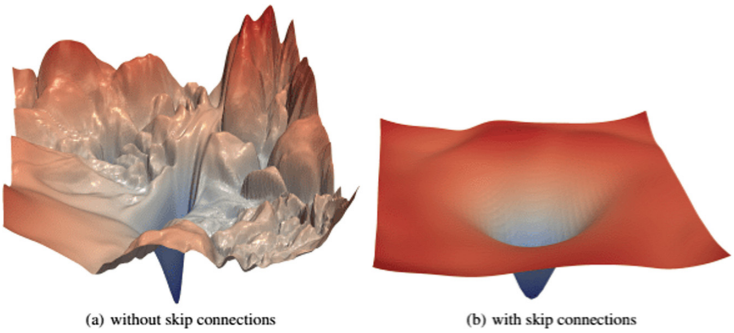

Fig. 2. Loss surface of a ResNet-56 without (left) and with (right) skip connections [18].

apparently limited impact on the algorithm, the choice of an initialization function can lead the algorithm to find a mostly linear neural network, or one with multiple skip connections, as illustrated in Fig. 3.

The first initialization considered is Glorot, proposed in 2010 and designed for DNNs with symmetric activation functions such as *tanh* and *softsign* [5,11]. It draws weights from a random uniform distribution such as:

$$W \sim \mathcal{U}(-\sqrt{\frac{6}{f_{in} + f_{out}}}, \sqrt{\frac{6}{f_{in} + f_{out}}}), \tag{1}$$

where f_{in} and f_{out} are the input and output sizes of the layer [5,20]. It also has a normal form, with $\mathrm{N}(0, \mathrm{std}^2)$ [21], where:

$$std = \sqrt{\frac{2}{f_{in} + f_{out}}}. \tag{2}$$

As pointed out by Goodfellow et al. [4], the formula is derived on the assumption that the network consists only of a chain of matrix multiplications, with no nonlinear activations.

He initialization, in contrast, is designed for non-symmetric activation functions such as ReLU. The weights in each layer are generated to approximate the derivative of the activation function from 0 to 1 [6,11]. Its uniform formula is as follows [6,22]:

$$W \sim \mathcal{U}(-\sqrt{\frac{6}{f_{in}}}, \sqrt{\frac{6}{f_{in}}}). \tag{3}$$

Similar to Glorot, the He function also contains a normal distribution form, $\mathcal{N}(0, \mathrm{std}^2)$ [23], where:

$$std = \sqrt{\frac{2}{f_{in}}}. \tag{4}$$

Although we have seen the emergence of non-monotonic activation functions as alternatives to new directions with the rise of functions such as swish [24] and mish [25], studies are still placed considering Glorot and He initializers. For the scope of this work, let us consider the normal and uniform versions of both Glorot and He.

As both functions are the top choice used for initiate a DNN, those functions are tested with CoDeepNEAT, alongside Random initialization. In the end, six activations are used in this paper: Glorot Normal, Glorot Uniform, He Normal, He Uniform, Random Normal and Random Uniform.

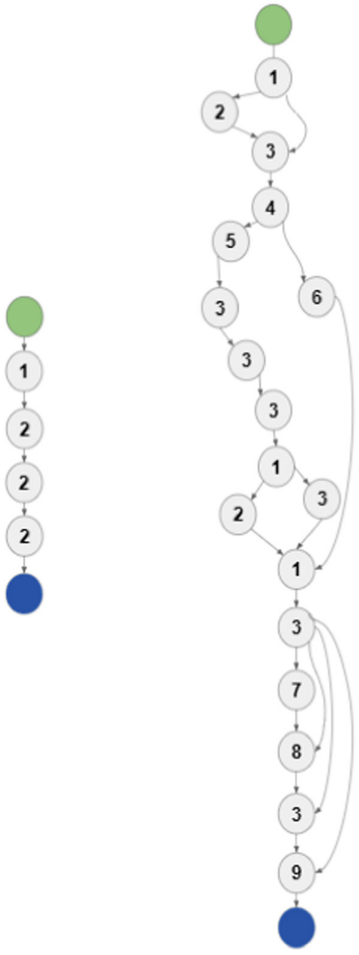

Fig. 3. Overall structure of two different networks evolved with Glorot initialization (left) and He (right). Each node represents a layer, and the different nodes represent different modules. Notice the linearity of the network on the left and the presence of skip connections on the right.

The performance between the weights initialization functions with different activation functions also needs to be addressed. Here the Hyperbolic Tangent (TanH, or tanh) and *Rectified Linear Unit* (ReLU, or relu) functions were chosen, both non-linear and commonly used as standard functions.

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{5}$$

$$\text{relu}(x) = \max(0, x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geqslant 0 \end{cases} \tag{6}$$

4 Experiments and Discussion

Our short-term, medium-term, and long-term experiments were based on and compared against two baselines. The first, henceforth named B1, is the original work that presented the CoDeepNEAT [12], and the second (B2), is the work of Boher et al. [14]. The striking aspect of B1 is the very large search space of the coevolutionary algorithm, as discussed in the previous section, whereas B2 has a much smaller search space. This is because the objective of Bohrer et al. was to reproduce the original experiments of the [12] with limited resources. Instead of multiple GPUs, they used a single CPU with only 30 GB RAM.

Our short-term experiments (ST) were inspired by both baselines, but with shorter training episodes than both of them. The sizes of the populations were similar to B2, but we kept the possible DNNs hyperparameters closer to B1. We also employed only a subset of the training data, and we reduced the duration of the coevolutionary algorithm even further (4 generations rather than 40). For medium-term (MT) and long-term (LT), we experimented with increasing the duration of the experiment, and we trained the individuals for more epochs. The experiments configurations are summarized in Table 1.

We compare the most commonly used initialization functions in Deep Learning: He, Glorot and Random. Each of them has a uniform and a normal variant, which determines the distribution it will draw weights from. For the sort-term experiments, we evaluate the CoDeepNEAT algorithm on two data sets (CIFAR-10 and MNIST) with five hold-out iterations. However, as medium-term and long-term experiments take substantially longer to complete, we had to limit the experiments to a single hold-out on the CIFAR-10 data set.

The short-term results are presented in Tables 2 and 3. Table 2 gives the accuracy of the best individual trained after the last generation on each of the five hold-out runs. Table 3 gives the mean and standard deviations.

First, let us discuss the experiments performed with the ReLU activation. We note that, in all runs, our results are better than the second baseline for the MNIST data set. The worst short-term result for MNIST was 99%, whereas B2 reported 92%. On CIFAR-10, the short-term results had a larger standard deviation. On average, the networks found with He initialization were similar to the baseline, while the individuals evolved with Glorot performed slightly

Table 1. Evolutionary and topological hyperparameters. B1 and B2 are baselines. ST, MT, and LT stand for short, medium, and long terms. B1 and B2 accuracies come from their sources. For ST, MT, and LT we report the lowest average obtained with ReLU and either Glorot or He in our experiments.

Parameters	B1 [12]	B2 [14]	ST	MT	LT
Generations	75	40	4	25	75
DNNs population	100	10	10	100	100
Blueprints population	25	10	10	25	25
Modules population	45	30	30	45	45
Epochs during evolution	8	4	5	10	10
Epochs in final training	300	40	100	150	150
Data used during evolution	100%	40%	40%	100%	100%
Filters	[32, 256]	[16, 48]	[32, 256]		
Kernel size	{1, 3}	{1, 3, 5}	{1, 3, 5}		
Dropout rate	[0, 0.7]	[0, 0.5]	[0, 0.5]		
Max pooling	{Yes, No}	{No}	{Yes, No}		
Batch normalization	{No}	{No}	{Yes, No}		
MNIST accuracy	–	92%	98.9%	–	–
CIFAR-10 accuracy	92.7%	77%	74.9%	80.1%	84.5%

better. Notice that the main difference from our short-term experiment to B2 was the larger search space for number of filters, and the possibility of employing max pooling and batch normalization. This is suggestive that running the DNE algorithm for longer generations is less important than providing the algorithm with more flexible modules.

Next, we repeated the short-term experiments with the hyperbolic tangent activation function (tanh). Again, the experiments were performed on CIFAR-10 and MNIST, and repeated a hold-out partitioning scheme five times. In both cases, the experiments were executed on single-CPU system with 15.5 GiB of RAM and a 4 GB GPU (GeForce GTX 1650/PCIe/SSE2). The total run time was 67 h. Thus, considering 2 activation functions, 4 initialization methods and 2 data sets, the total combinations add up to 16 distinct experimental settings, each repeated five times, so each experiment took on average 50 min to complete.

The average accuracy and its standard deviation are in Table 3. The performance of the individuals found with tanh was severely hindered when compared to the individuals evolved with ReLU. Still, with the exception of Glorot Normal, the results were better than B2. Once again, this suggests that it is possible to achieve decent results when the DNE is tuned to find "fast-learners", DNNs which can be trained with few epochs during the evolutionary process.

For the remaining experiments, we focused only on the CIFAR-10 data set and ReLU activation function.

Table 2. Short-term experiments with Glorot and He initialization functions (ReLU activation).

CIFAR-10			MNIST		
Initialization	*Run*	Accuracy	Initialization	*Run*	Accuracy
Glorot normal	1	0.7681	Glorot normal	1	0.9872
Glorot normal	2	0.7861	Glorot normal	2	0.9893
Glorot normal	3	**0.8057**	Glorot normal	3	0.9900
Glorot normal	4	0.8034	Glorot normal	4	0.9893
Glorot normal	5	0.7995	Glorot normal	5	**0.9903**
Glorot uniform	1	**0.8462**	Glorot uniform	1	0.9890
Glorot uniform	2	0.7188	Glorot uniform	2	0.9908
Glorot uniform	3	0.8118	Glorot uniform	3	0.9876
Glorot uniform	4	0.7654	Glorot uniform	4	**0.9914**
Glorot uniform	5	0.8180	Glorot uniform	5	0.9881
He normal	1	0.7798	He normal	1	0.9879
He normal	2	0.6894	He normal	2	0.9910
He normal	3	0.7159	He normal	3	0.9870
He normal	4	0.7706	He normal	4	**0.9921**
He normal	5	**0.7919**	He normal	5	0.9899
He uniform	1	0.7743	He uniform	1	0.9902
He uniform	2	0.7767	He uniform	2	**0.9923**
He uniform	3	0.7324	He uniform	3	0.9903
He uniform	4	0.7695	He uniform	4	0.9900
He uniform	5	**0.7998**	He uniform	5	0.9870

Table 3. Short-term accuracy with Glorot and He initialization functions with linear and non-linear activation functions.

Data set	Initialization	Activation	Mean accuracy	Std. dev.
MNIST	Glorot normal	tanh	0.9089	±0.1116
MNIST	Glorot uniform	tanh	0.9723	±0.0046
MNIST	He normal	tanh	0.9698	±0.0045
MNIST	He uniform	tanh	0.9733	±0.0026
CIFAR-10	Glorot normal	tanh	0.4532	±0.0547
CIFAR-10	Glorot uniform	tanh	0.5487	±0.0658
CIFAR-10	He normal	tanh	0.5143	±0.1237
CIFAR-10	He uniform	tanh	0.5172	±0.0777
MNIST	Glorot normal	ReLU	0.9892	±0.0010
MNIST	Glorot uniform	ReLU	0.9893	±0.0014
MNIST	He normal	ReLU	0.9895	±0.0018
MNIST	He uniform	ReLU	0.9899	±0.0016
CIFAR-10	Glorot normal	ReLU	0.7925	±0.0139
CIFAR-10	Glorot uniform	ReLU	0.7920	±0.0448
CIFAR-10	He normal	ReLU	0.7495	±0.0397
CIFAR-10	He uniform	ReLU	0.7705	±0.0217

The medium-term in this section were carried out in order to contemplate a compromise proposal between the two *baselines* used, since Boher et al. [14] used a very small amount of computational power when compared to what was used in the original CoDeepNEAT paper (2,000,000 CPUs and 5,000 GPUs). We ran the medium-term experiments on the same architecture as the short-term (a single 4 GB GPU). This time, however, each experiment took approximately 32 h in lieu of the average 50 min observed in the short-term experiments. Therefore, we limited the medium-term experiments to a single hold-out experiment.

The initialization functions were both the uniform and normal variants of He and Glorot, and we also consider a random initialization, which assigns weight values to the neurons without taking into consideration neither the activation function, nor the sizes of the input or the output. The results are shown in Table 4. In addition to the accuracy, we also report the number of trainable parameters of the best individual and whether that individual contains residual connections or not.

Table 4. Best individuals from experiments in medium-term for each evolutionary process.

Data set	Initialization	Final individual	Accuracy	Parameters
CIFAR-10	He uniform	Residual	0.8787	≈3.33 M
CIFAR-10	He normal	Residual	0.8336	≈1.25 M
CIFAR-10	Glorot uniform	Sequencial	0.8011	≈2.25 M
CIFAR-10	Glorot normal	Sequencial	0.8581	≈1.15 M
CIFAR-10	Random uniform	Sequencial	0.7380	≈1.06 M
CIFAR-10	Random normal	Sequencial	0.7271	≈2.03 M

Considering the greater number of generations, the DNNs assembled by CoDeepNEAT may obtain some residual connection due to mutations in their blueprints. However, we noticed that only the He-like initialization were able to produce the best individual with some residual connection, while all the others converged to an individual whose architecture was fully sequential (total absence of skip connections). It is important to mention that, during the evolutionary process of all, individuals with some degree of residuality were found in all cases, but only the architectures initialized with He Uniform or He Normal managed to create individuals with residual architectures that surpassed the performance of individuals with non-residual architectures during their respective evolutionary processes.

Although one of the He initializations had the best result, as seen in Table 4, with 87.87% accuracy, the Glorot Normal initialization achieved a similar result, 85.81%, with approximately 34.72% fewer floating-point operations and 34.53% less total parameters, which is equivalent to the total number of parameters to perform an inference on the model. So He Uniform got the best result, but the

Glorot Normal initializer is just as competitive with less computing resources required. A reasonable explanation for this fact is that the high frequency of mutations allows the development of residual connections, but a large number of generations may be required to produce individuals that are competitive. So much so that Miikkulainen et al. state in [12] that it was only after the 70^{th} generation that the accuracy of the individuals in their experiments stabilized.

In order to explore this possibility, we executed long-term experiments, with three times as many generations as the medium-term. However, as the neural networks become more complex over the generations, the total time to finish was substantially greater: 40 days per experiment (∼920 h), when compared to the 32 h required to finish each medium-term experiment. Only 1 hold-out was considered for Glorot Normal and He Uniform, which were the most accurate initializations in the medium-term experiments. The results are presented in Table 5.

Table 5. Best individuals from experiments in long-term for each evolutionary process.

Data set	Initialization	Final individual	Accuracy	Parameters
CIFAR-10	He uniform	Residual	0.8991	≈10.23 M
CIFAR-10	Glorot normal	Sequencial	0.8454	≈0.47 M

Similarly to the previous results, the best individual evolved with He initialization achieved better accuracy (89.91%) than the individual evolved with Glorot (85.54%). However, the DNN found with Glorot was completely linear, as shown in Fig. 3 (left). In comparison, the DNN found with He was rich in skip connections.

5 Conclusion

In this paper we considered the influence of different experimental settings of a coevolutionary algorithm We performed experiments with coevolution of Deep NeuroEvolution of Augmenting Topologies (CoDeepNEAT) on two benchmark data sets, on three different scenarios that involve increasingly more complex search spaces. We compared the results obtained with two popular initialization functions, He and Glorot, as well as a random-initialization strategy serving as baseline. In addition, for short-term experiments we also compared linear and non-linear activation functions.

In the initial experiments, focusing in short-term parameters, ReLU activation outperformed the results obtained by tanh in all experiments. While the results are somewhat below start-of-the-art performances, they were above 99.20% in the MNIST dataset and 89.91% in the CIFAR-10 dataset, outperforming the baseline study of [14] which attempted to run the CoDeepNEAT with

limited computational resources. Our result shows that it may be more important to fine-tune the coevolutionary algorithm hyperparameters and attempt to evolve "fast-learners" in fewer generations than perform longer experiments.

It is important to note that the best result obtained for CIFAR-10 (89.91%) was derived from long-term experiments. Furthermore, we draw attention to a curious fact: considering the universe of medium and long-term experiments, all the best individuals evolved with Glorot (both Uniform and Normal variants) did not contain residual connections, while all of the best He individuals (Uniform or Normal) had multiple residual connections and were deeper. Considering the importance of residuality in neural architecture design, there seems to be evidence to suggest an investigation into this greater capacity of He initializations to facilitate the development of efficient residual architectures in short-term evolution processes.

References

1. Ma, Y., Xie, Y.: Evolutionary neural networks for deep learning: a review. Int. J. Mach. Learn. Cybern. (2022). https://doi.org/10.1007/s13042-022-01578-8
2. Kumar, S. K.: On weight initialization in deep neural networks. In: arXiv preprint arXiv:1704.08863 (2017)
3. Initializing neural networks. https://www.deeplearning.ai/ai-notes/initialization/. Accessed 12 June 2022
4. Goodfellow, I.J., Bengio, Y., Courville, A.: Deep Learning, 1st edn. Cambridge (2016)
5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256. JMLR Workshop and Conference Proceedings, Sardinia (2010)
6. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034. IEEE, Santiago (2010)
7. Koutnik, J., Gomez, F., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 619–626. ACM, Portland (2010)
8. Togelius, J., Gomez, F., Schmidhuber, J.: Learning what to ignore: memetic climbing in topology and weight space. In: Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pp. 3274–3281. IEEE, Hong Kong (2008)
9. Okada, H., Wada, T., Yamashita, A., Matsue, T.: Interval-valued evolution strategy for evolving neural networks with interval weights and biases. In: Proceedings of the International Conference on Soft Computing and Intelligent Systems, and the 13th International Symposium on Advanced Intelligence Systems, pp. 2056–2060. IEEE, Kobe (2012)
10. Desell, T.: Accelerating the evolution of convolutional neural networks with node-level mutations and epigenetic weight initialization. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 157–158. IEEE, Kyoto (2018)

11. Lyu, Z., ElSaid, A., Karns, J., Mkaouer, M., Desell, T.: An experimental study of weight initialization and weight inheritance effects on neuroevolution. In: Proceedings of Applications of Evolutionary Computation: 24th International Conference. ACM, Seville (2021)
12. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A.: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312 (2019)
13. Papavasileiou, E., Cornelis, J., Jansen, B.: A systematic literature review of the successors of 'NeuroEvolution of augmenting topologies'. Evol. Comput. **29**, 1–73 (2020)
14. Bohrer, J.S., Grisci, B.I., Dorn, M.: Neuroevolution of neural network architectures using CoDeepNEAT and Keras. In: arXiv preprint arXiv:2002.04634 (2020)
15. Zhou, X., Li, X., Hu, K., Zhang, Y., Chen, Z., Gao, X.: ERV-Net: an efficient 3D residual neural network for brain tumor segmentation. Expert Syst. Appl. **170**, 114566 (2021)
16. Dogan, S, et al.: Automated accurate fire detection system using ensemble pretrained residual network. Expert Syst. Appl. **203**, 117407 (2022)
17. Hoorali, F., Khosravi, H., Moradi, B.: IRUNet for medical image segmentation. Expert Syst. Appl. **191**, 116399 (2022)
18. Li, H., Xu, Z., Tyalor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: Advances in Neural Information Processing Systems (2018)
19. Intuitive Explanation of Skip Connections in Deep Learning. https://theaisummer.com/skip-connections/. Accessed 12 June 2022
20. Keras Documentation - Glorot Uniform. https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotUniform. Accessed 10 July 2022
21. Keras Documentation - Glorot Normal. https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotNormal. Accessed 10 July 2022
22. Keras Documentation - He Uniform. https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeUniform. Accessed 10 July 2022
23. Keras Documentation - He Normal. https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeNormal. Accessed 10 July 2022
24. Searching for activation functions. https://arxiv.org/abs/1710.05941. Accessed 12 June 2022
25. Mish: A self regularized non-monotonic neural activation function. https://arxiv.org/abs/1908.08681. Accessed 12 June 2022