Evolutionary search algorithm for learning activation function of an artificial neural network

Viacheslav Yurshin*

Siberian Federal University, Institute of Space and Information Technologies, Krasnoyarsk, Russian Federation

Abstract. Neural networks require careful selection of activation functions to optimize performance. Traditional methods of choosing activation functions through trial and error are time-consuming and resource-intensive. This paper presents a novel approach to automatically design activation functions for artificial neural networks using genetic programming combined with gradient descent. The proposed method aims to enhance the efficiency of the search process for optimal activation functions. Our algorithm employs genetic programming to evolve the general form of activation functions, while gradient descent optimizes their parameters during network training. This hybrid approach allows for the exploration of a wide range of potential activation functions tailored to specific tasks and network architectures. The method was evaluated on three datasets from the KEEL repository: Iris, Titanic, and Phoneme. The results demonstrate the algorithm's ability to generate and optimize custom activation functions, although improvements in network accuracy were not observed in this initial study. This work contributes to the ongoing research in neural network optimization and opens avenues for further investigation into the automatic design of activation functions.

1 Introduction

This paper will focus on the search for activation functions for artificial neural networks. Neural networks consist of layers of various combinations of linear and nonlinear functions. There are a huge number of activation functions such as ReLU, ELU, Mish, etc. The selection and design of original activation functions continues to be an active area of research, and there are many activation functions, each with different properties [1]. In certain settings and depending on the task, the accuracy results can vary greatly.

The design and selection are driven by the evolving understanding of what makes an activation function effective. The simplest are linear activation functions. But it has a big drawback, at the output the neural network gives a linear function of the input data and it turns out there is no sense in a multilayer network. In order to introduce nonlinearity, activation functions such as Sigmoid and Tangent began to be used. But these functions also have a drawback known as the vanishing gradient problem [2]. When calculating the derivative of the function, the value turns out to be extremely small, this problem is

* vy.yurshin@gmail.com

aggravated by increasing layers. The most popular is the ReLU (Rectified Linear Unit) [3] function. It has advantages over the Sigmoid and Tangent, such as the simplicity of calculating the derivative and reducing computational costs. But this activation function also has disadvantages, such as not using negative values, limited nonlinearity and unlimited output. In order to solve these problems, the Leaky ReLU function is used (ReLU with "leakage"), it allows you to transmit a small amount of information about the gradient when the input data is negative. There is also an activation function ELU (Exponential Linear Unit) that allows you to solve the problem of gradient vanishing. These two completely different activation functions have seemingly contradictory properties, but each of them turned out to be more effective than ReLU in solving various problems. In addition, there are often complex interactions between the activation function and other neural network design choices, further complicating the choice of an appropriate activation function for a given task. For example, studies [4] caution that when training using the Swish activation function, the scale parameter should be specified in batch normalization. Hendrycks and Gimpel [5] suggested using a momentum optimizer when using GELU under a Gaussian distribution. Next, a modification of elimination, called alpha elimination, was introduced [6] for use with SELU. These results show that significant gains are possible by properly designing the activation function for the network and the task, but doing so manually is difficult.

This study introduces an innovative approach to automatically generate activation functions for neural networks. The proposed technique leverages genetic programming, a paradigm that mimics biological evolution to develop computer programs tailored to specific tasks. In contrast to earlier research, this work emphasizes the discovery of parameterized activation functions through an evolutionary process. The algorithm employs a two-step strategy: first, it uses evolution to determine the overall structure of the function, and then applies gradient descent to fine-tune the function's parameters during neural network training.

The method combines the exploratory power of genetic algorithms with the optimization capabilities of gradient-based techniques. By evolving the general form of activation functions, the approach can potentially uncover novel and effective designs that might be overlooked by traditional manual selection methods. The subsequent parameter optimization allows these evolved functions to be further refined, potentially leading to improved neural network performance across various tasks.

2 Description of the algorithm

The general scheme of the developed algorithm is presented in Figure 1.

Usually, the selection of neural network parameters is done randomly, then the efficiency of the neural network is tested and the parameters are changed using certain methods. This is repeated until the neural network is trained.

In this work, a fully connected artificial neural network is implemented, allowing you to change:

- The number of layers of the artificial neural network.
- The number of neurons in each layer.
- The type of activation function obtained using GP.

The general scheme of the developed algorithm is illustrated in Figure 1, which provides a comprehensive overview of the evolutionary search process for optimizing activation functions in artificial neural networks. The diagram outlines a multi-step approach that combines genetic programming with gradient descent optimization.

The algorithm begins with the initialization of a population of random activation functions. These functions are then evaluated within a neural network to assess their performance. Based on this evaluation, the algorithm selects the most promising candidates

for reproduction. The reproduction process involves genetic operations such as crossover and mutation, which generate new activation functions by combining and altering existing ones.

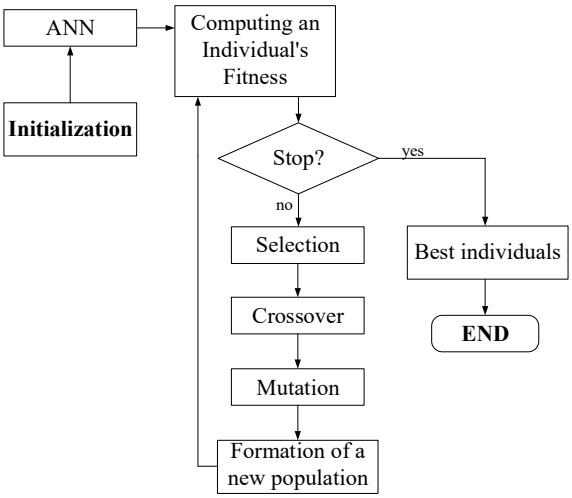


Fig. 1. Algorithm diagram.

These newly created functions undergo parameterization, where their specific parameters are fine-tuned using gradient descent during the neural network training process. This hybrid approach allows for both the exploration of novel function structures through evolution and the optimization of their parameters through traditional machine learning techniques.

The algorithm then evaluates the fitness of these new functions by testing them in the neural network. Functions that demonstrate improved performance are retained in the population, replacing older or less effective ones. This cycle of selection, reproduction, parameterization, and evaluation continues for a predetermined number of generations or until a satisfactory activation function is discovered.

Throughout this process, the algorithm maintains a balance between exploration of new function designs and exploitation of promising candidates, aiming to discover activation functions that enhance the neural network's performance on specific tasks.

2.1 Description of genetic programming

Table 1 in the paper presents a comprehensive set of operators used in the genetic programming approach for designing activation functions. These operators serve as the building blocks for constructing complex activation functions through evolutionary processes.

Table 1. Operators.

Operators						
x^2	+	-	*	/	sin	cos

Each operator in the table represents a node in the computation graph of the activation function. By combining these operators in various ways, the genetic programming algorithm can explore a wide range of potential activation functions, from simple linear combinations to highly nonlinear and complex functions. The inclusion of both basic arithmetic and

nonlinear operators allows the algorithm to generate diverse activation functions that can potentially capture different aspects of the data relationships. This diversity is crucial for finding optimal activation functions for specific neural network architectures and tasks.

At the initial stage of operation, the algorithm initializes a population of N random activation functions, consisting of the presented operators. An example of a function is presented in Table 2.

Individuals of the first generation are generated randomly using a uniform distribution.

Crossover. A random subtree of one individual is taken and crossed with a subtree of another individual.

Mutation replaces each operator in the graph with another operator from the proposed ones. Although each node of the graph changes, the overall structure of the computation graph remains the same.

The evolutionary process for discovering activation functions begins with the creation of a diverse set of random functions. These initial functions are parameterized and evaluated based on their performance. The algorithm then employs a selection mechanism to choose N functions from the existing population, allowing for potential repetition.

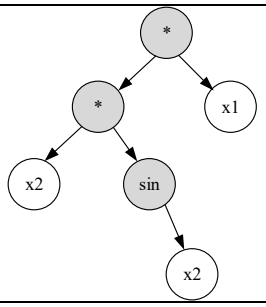
From this selection, the function demonstrating the highest accuracy is designated as the parent. This parent function undergoes mutation to produce a new offspring activation function. The newly created function is then parameterized and its fitness is assessed. The fitness evaluation involves using the function in a neural network to classify data, with the classification accuracy serving as the fitness metric.

Once evaluated, the new activation function is integrated into the population, while the oldest function is removed to maintain a constant population size. This cycle of selection, mutation, evaluation, and population update continues iteratively. The process persists until all potential activation functions have been examined and evaluated.

Throughout this evolutionary journey, the algorithm keeps track of the best-performing function across all generations. Upon completion, this top-performing activation function is selected as the final output. To validate its effectiveness, the chosen function undergoes testing within a fully connected neural network architecture developed specifically for this purpose.

This approach leverages the power of evolutionary algorithms to explore a vast space of potential activation functions, potentially uncovering novel and effective designs that might be overlooked by traditional manual selection methods

Table 2. Coded function.

Coded function	**Graph shape**
((x2 * sin(x2)) * x1)	

The algorithm employs a selective retention strategy for activation functions. When a newly generated function underperforms compared to its predecessor in terms of fitness score, it is eliminated from consideration. While these underperforming functions are

excluded from the population, they are still accounted for in the overall tally of evaluated functions for each neural network architecture.

This stringent selection process serves as a quality assurance measure, ensuring that only high-performing activation functions persist in the population. By culling inferior candidates, the evolutionary process becomes more streamlined and efficient, concentrating computational resources on refining and exploring the most promising functional forms. This approach effectively narrows the search space, potentially accelerating the discovery of optimal activation functions for specific neural network tasks.

3 Testing the neural network

A fully connected artificial neural network was used for testing, into which the obtained activation functions were embedded. It was used to classify data taken from the KEEL website [7]. The data sets selected were “Iris” (4 features, 3 classes, 150 examples), “Titanic” (3 features, 2 classes, 2201 examples), “Phoneme” (5 features, 2 classes, 5404 examples).

Table 3 presents the classification accuracies and encoded activation functions for 50 generations of the algorithm.

Table 3. Test results.

Sample	Received function
Iris	(((x cos (y)) - (y * (y))) sin ((x cos (x)) + (x sin (x))))
Phoneme	(((x / x) * (x cos (y))) + ((y * y) + (y + x)))
Titanic	(((y * x) / (y * x)) sin ((x - y) sin (x / y)))

4 Conclusion

This study presented an innovative algorithm for the automatic search and design of activation functions in artificial neural networks using genetic programming combined with gradient descent optimization. While the proposed method successfully generated and optimized custom activation functions, it did not yield significant improvements in neural network accuracy for the tested datasets.

Despite the lack of immediate performance gains, this research contributes valuable insights to the field of neural network optimization. The developed algorithm demonstrates the feasibility of automating the activation function design process, potentially reducing the time and resources typically required for manual selection and tuning.

The challenges encountered in this study highlight the complexity of optimizing neural network architectures and underscore the need for continued research in this area. Future work could explore several promising directions, such as incorporating existing high-performing activation functions as initial seeds in the genetic programming process, expanding the range of datasets and neural network architectures tested, refining the fitness evaluation criteria, and investigating potential synergies between evolved activation functions and other neural network components.

While the current implementation did not increase neural network accuracy, the approach presented here opens new avenues for research in automated neural network design. As the field continues to evolve, techniques like the one proposed in this paper may play a crucial role in developing more efficient and effective neural network architectures.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (grant No. 075-15-2022-1121).

References

1. A. V. Sozykin, Vestnik of SUSU. Series: Comput. Mat. Inform., **6(3),** 28-59 (2017).
2. S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *A Field Guide to Dynamical Recurrent Networks*, J. F. Kolen and S. C. Kremer, eds. (Wiley-IEEE Press, 2003).
3. V. Nair, G.E. Hinton, *Rectified linear units improve restricted boltzmann machines* in: Proceedings of the 27th International Conference on Machine Learning, pp. 807– 814 (2010).
4. P. Ramachandran, B. Zoph, Q. V. Le. *Searching for activation functions. In 6th International conference on learning representations* in: ICLR 2018, Vancouver, BC, Canada, April 30 - May 3 (2018).
5. D. Hendrycks, K. Gimpel, *Gaussian error linear units (gelus).* arxiv: 1606.08415 (2016) https://doi.org/10.48550/arXiv.1606.08415
6. G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, *Self-normalizing neural networks. In Advances in neural information processing systems,* arXiv:1706.02515 (2017) https://doi.org/10.48550/arXiv.1706.02515
7. J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, F. Herrera, Soft Comput., **13(3),** 307-318 (2009).