Журнал обчислювальної та прикладної математики **2025, № 1** Journal of Numerical & Applied Mathematics

УДК 004.032.26
MSC 37D45, 68T07

DERIVATIVE-FREE OPTIMIZATION FOR CUSTOM LOSS FUNCTIONS

O. S. Maistrenko

Faculty of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, E-mail: o.maistrenko@knu.ua, ORCID: 0009-0003-3464-7777

ОПТИМІЗАЦІЯ БЕЗ ПОХІДНИХ ДЛЯ СПЕЦІАЛІЗОВАНИХ ФУНКЦІЙ ВТРАТ

О. С. Майстренко

Факультет комп'ютерних наук та кібернетики, Київський національний університет імені Тараса Шевченка, Київ, Україна, E-mail: o.maistrenko@knu.ua, ORCID: 0009-0003-3464-7777

Abstract. Derivative-free optimization (DFO) has emerged as a powerful technique for solving optimization problems where the gradient of the objective function is either unavailable, expensive to compute, or non-smooth. This article explores the application of DFO methods to optimize custom loss functions in machine learning and other fields. The paper also highlights the challenges and potential improvements in the current DFO approaches, offering insights for further research and practical applications.
Keywords: Derivative-free optimization, machine learning, black-box optimization.

Анотація. Оптимізація без похідних (DFO) набула значної популярності як потужний підхід до розв'язання задач оптимізації, у яких похідна цільової функції є недоступною, надто складною або дорогою для обчислення, або ж сама функція не є гладкою. Ця стаття присвячена дослідженню застосування методів оптимізації без похідних до налаштування спеціалізованих функцій втрат у машинному навчанні та суміжних галузях. Окрему увагу приділено існуючим викликам, з якими стикаються сучасні DFO-методи, зокрема — проблемам масштабованості, вибору гіперпараметрів, ефективності пошуку в просторах високої розмірності, а також адаптації до шумних або непередбачуваних функцій. У статті також обговорюються перспективи вдосконалення цих методів, включаючи інтеграцію з евристичними підходами, метаевристиками, а також можливості поєднання з методами навчання з підкріпленням або байєсівської оптимізації.
Ключові слова: оптимізація без похідних, машинне навчання, оптимізація «чорної скриньки».

1. Introduction

Neural networks have become the backbone of modern artificial intelligence, powering applications in computer vision, natural language processing (NLP), reinforcement learning (RL), and beyond. The success of deep learning is largely attributed to the availability of large-scale datasets, powerful computational resources, and sophisticated optimization techniques that allow neural networks to learn complex patterns from data. Most commonly, gradient-based optimization methods such as stochastic gradient descent (SGD) and its numerous variants — including Adam, RMSprop, and Adagrad — are used to update network parameters by leveraging the gradient of the loss function with respect to the model weights. These techniques have proven highly effective when dealing with standard loss functions that are differentiable and smooth, such as mean squared error (MSE) for regression tasks or cross-entropy loss for classification problems.

However, in many real-world applications, the use of standard loss functions is insufficient or even inappropriate. Custom loss functions may be required when dealing with domain-specific objectives, non-traditional evaluation metrics, or real-world constraints that are difficult to express in a differentiable form. Examples include loss functions that depend on ranking metrics (such as mean average precision in information retrieval), loss functions involving discrete variables (such as edit distance in text generation), or loss functions that integrate external black-box evaluations (such as real-world reinforcement learning rewards or adversarial robustness measures). In such cases, computing exact gradients can be challenging or even impossible, rendering traditional gradient-based methods ineffective.

Derivative-free optimization methods offer a promising alternative by optimizing neural networks without requiring explicit gradient information. Unlike gradient-based approaches, which rely on the chain rule and backpropagation, DFO methods operate by evaluating different candidate solutions and updating model parameters based on function values alone. These methods are particularly useful when dealing with non-differentiable, noisy, or black-box objective functions. Various classes of DFO techniques exist, including evolutionary algorithms (such as genetic algorithms and covariance matrix adaptation evolution strategy), Bayesian optimization, Nelder-Mead simplex methods, and direct search techniques (such as pattern search and Powell's method). Each of these methods has its own strengths and weaknesses, making them suitable for different types of optimization problems [1, 2].

The use of DFO for training neural networks presents both opportunities and challenges. On the one hand, DFO methods can optimize models in scenarios where gradient-based methods fail, such as when dealing with loss functions defined by simulation-based or experimental evaluations. They can also be highly effective in optimizing hyperparameters or architectures alongside network weights [3]. On the other hand, DFO methods tend to require more function evaluations than gradient-based methods, making them computationally expensive for high-dimensional problems. Additionally, since DFO techniques often rely on heuristic or probabilistic search strategies, they may exhibit slower

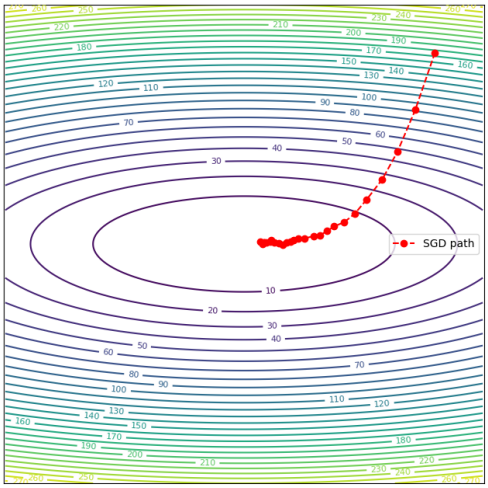

FIGURE 1. Example of gradient-based optimization.

convergence compared to gradient-based alternatives in smooth, well-behaved loss landscapes [4].

In this article, we explore the application of derivative-free optimization techniques for training neural networks with custom loss functions. We provide an overview of the key DFO approaches, discuss their theoretical foundations, and analyze their practical applicability. Through empirical experiment, we demonstrate the effectiveness of these methods in optimizing neural networks when standard gradient-based approaches are infeasible. By examining real-world use case, we highlight the scenarios in which DFO can offer superior performance, as well as its limitations and trade-offs. Our goal is to provide a comprehensive understanding of how derivative-free optimization can be leveraged to extend the capabilities of neural network training beyond the confines of traditional gradient-based methods [5].

2. Optimization techniques' comparison

The process of optimizing neural networks is fundamental to achieving high performance in machine learning tasks, and the choice of optimization technique significantly impacts the efficiency, accuracy, and convergence speed of the model. Traditionally, gradient-based optimization methods have been the dominant approach, particularly in deep learning, where backpropagation enables efficient parameter updates by computing gradients of a loss function. However, when dealing with complex loss functions that are non-differentiable, noisy, or dependent on external black-box evaluations, DFO methods provide an alternative framework. Each optimization technique offers distinct advantages and drawbacks, making the choice of method crucial for specific applications [6].

Gradient-based optimization has long been the standard method for training neural networks due to its efficiency in large-scale problems with differentiable loss functions. The most common approach, stochastic gradient descent,

updates model parameters in the direction of the negative gradient of the loss function with respect to the weights.

Variants such as Adam, RMSprop, and Adagrad introduce adaptive learning rates, momentum, and other heuristics to enhance convergence and stability (see Figure 1).

These methods leverage backpropagation, a core algorithm that propagates errors backward through the network to compute exact gradients [7]. This approach works exceptionally well when the loss function is smooth and differentiable, as it ensures rapid convergence to an optimal or near-optimal solution. However, its effectiveness is contingent upon the existence of well-defined gradients. When the loss function contains discontinuities, non-differentiable components, or is defined through external processes such as simulations or adversarial interactions, gradient computation becomes infeasible, rendering traditional methods ineffective [8].

Another drawback of gradient-based methods is their tendency to converge to local minima, especially in highly non-convex optimization landscapes. Although deep neural networks often rely on heuristics such as batch normalization and learning rate schedules to escape poor local minima, there are cases where these techniques fall short, particularly when optimizing irregular or highly constrained objectives [9]. Furthermore, gradient-based methods can struggle when dealing with noisy objectives, as small perturbations in the function evaluations can lead to unstable updates. These limitations necessitate alternative approaches, particularly in scenarios where gradients cannot be computed analytically or efficiently.

Derivative-free optimization methods circumvent the reliance on gradients by directly evaluating function values to explore the parameter space. One of the most powerful categories of such techniques is evolutionary algorithms, which take inspiration from natural selection and biological evolution. Methods such as Genetic Algorithms (GA) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) operate by evolving a population of candidate solutions over multiple generations. These algorithms utilize mechanisms such as mutation, crossover, and selection to refine solutions progressively. In a comparative study [10], CMA-ES achieved higher average performance than Adam and SGD in optimizing non-differentiable benchmark functions such as Rastrigin and Ackley, though at the cost of significantly more function evaluations (see Table 1). Unlike gradient-based optimization, which follows a deterministic path dictated by gradients, evolutionary algorithms employ stochastic exploration, making them highly effective in avoiding local minima and handling complex loss landscapes [11].

One major advantage of evolutionary algorithms is their ability to optimize highly non-linear, discontinuous, and black-box loss functions. They can efficiently search large parameter spaces without requiring gradient information, making them suitable for problems where loss functions are defined by simulations, rule-based evaluations, or external feedback loops. Additionally, evolutionary methods are inherently parallelizable, as multiple candidate solutions can be evaluated simultaneously, enabling efficient computation on modern

Method	Differentiability Required	Convergence Speed	Robustness to Noise	Scalability	Sample Efficiency
SGD/Adam	Yes	High	Low	High	High
CMA-ES	No	Moderate	High	Moderate	Low
Bayesian Opt.	No	Low	High	Low	High
Nelder-Mead	No	Low	Moderate	Low	Low

TABLE 1. Summary of Comparative Performance Across Optimizers.

distributed systems. However, this flexibility comes at a cost [12]. Evolutionary algorithms tend to require significantly more function evaluations than gradient-based methods, making them computationally expensive. As the search progresses through multiple generations, the number of required evaluations grows, leading to higher computational costs compared to SGD-based approaches. Furthermore, fine-tuning hyperparameters such as mutation rates and population sizes are non-trivial and often problem-specific, requiring careful experimentation to achieve optimal performance [13].

Another powerful derivative-free optimization technique is Bayesian optimization, which differs from evolutionary algorithms by constructing a probabilistic model of the objective function. Instead of randomly exploring the parameter space, Bayesian optimization strategically selects points to evaluate based on a balance between exploration and exploitation. This is achieved through the use of a surrogate model, typically a Gaussian Process (GP), which approximates the unknown loss function [14]. An acquisition function is then used to determine the next evaluation point by considering both regions of high uncertainty and promising areas where improvements are expected.

Bayesian optimization is particularly useful in situations where function evaluations are expensive, such as hyperparameter tuning for neural networks or optimization problems involving costly real-world experiments (see Figure 2). By intelligently selecting the most informative evaluation points, Bayesian methods can find near-optimal solutions with significantly fewer function evaluations compared to exhaustive search techniques. In experiments on tuning neural network hyperparameters, Bayesian optimization required up to $5\times$ fewer evaluations than random search to reach similar performance levels [15]. However, Bayesian optimization also has limitations. One of its primary drawbacks is its scalability; as the number of dimensions increases, the computational complexity of maintaining and updating the surrogate model grows exponentially, making it less practical for high-dimensional problems like deep neural network training [15]. Additionally, Bayesian methods rely on assumptions about the function's smoothness, which may not always hold in highly irregular optimization landscapes.

In contrast to both evolutionary algorithms and Bayesian optimization, the Nelder-Mead simplex method is a direct search technique that optimizes functions using a geometric simplex of points [16]. This algorithm iteratively modifies the simplex through operations such as reflection, expansion, contraction, and shrinkage, allowing it to navigate the optimization landscape without requiring

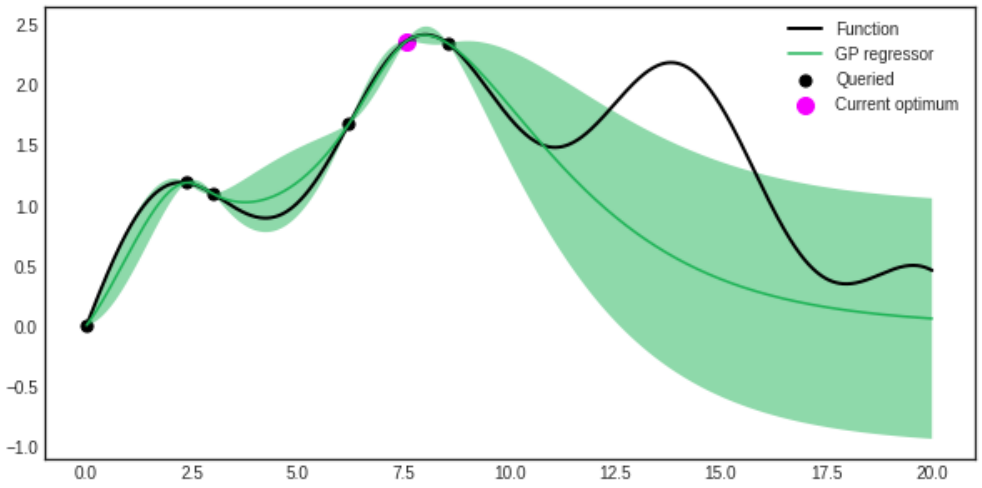

FIGURE 2. Example of Bayesian optimization algorithm.

gradients. The Nelder-Mead method is particularly effective for low-dimensional problems where function evaluations are relatively inexpensive. It has been widely used in engineering and applied sciences for optimizing smooth and unimodal functions.

Despite its simplicity, the Nelder-Mead method has notable limitations. It does not scale well to high-dimensional spaces, as the number of required function evaluations increases rapidly with the number of parameters [17]. Additionally, it can struggle with highly non-convex or discontinuous loss functions, often getting trapped in local minima. Due to these limitations, it is rarely used for large-scale neural network training but may still be useful in tuning smaller architectures or optimizing auxiliary parameters.

A related family of techniques includes pattern search and Powell's method, which systematically explores the parameter space by evaluating function values along specific directions. Powell's method, in particular, refines search directions iteratively to accelerate convergence. These methods share similar strengths and weaknesses with Nelder-Mead, being suitable for medium-scale problems but becoming inefficient in very high-dimensional spaces.

When comparing these optimization methods, several key factors determine their suitability for different problems. Gradient-based optimization remains the fastest and most efficient approach for large-scale deep learning tasks with differentiable loss functions, making it the default choice in most scenarios. However, in cases where gradients are unavailable or unreliable, derivative-free methods provide viable alternatives. Evolutionary algorithms are particularly effective for global optimization problems where avoiding local minima is critical, while Bayesian optimization is ideal for expensive function evaluations that require strategic sampling. Direct search methods like Nelder-Mead

and Powell's method, though simple, are best suited for small-scale or well-structured optimization problems [18, 19].

In practical applications, hybrid approaches often yield the best results. For example, evolutionary strategies can be used for coarse optimization before refining solutions with gradient-based methods. Similarly, Bayesian optimization can assist in tuning hyperparameters before employing SGD for final network training. The optimal choice of method ultimately depends on the problem's complexity, computational constraints, and the nature of the loss function being optimized.

3. Experiment with p-statistic

Next, we will examine the effectiveness of DFO in an experiment utilizing a custom loss function.

The Petunin statistic, or p-statistic, is a measure of closeness between samples, proposed by Yuriy Petunin. It is used to test the hypothesis that the distribution functions of two samples are equal.

In [20] two populations G and G' and the corresponding distribution functions F_G and $F_{G'}$ are considered.

Let $x = (x_1, x_2, ..., x_n) \in G$ and $x = (x'_1, x'_2, ..., x'_n) \in G'$, also $x_{(1)} \le x_{(2)} \le ... \le x_{(n)}$ and $x'_{(1)} \le x'_{(2)} \le ... \le x'_{(m)}$ be the corresponding ordinal statistics. Suppose that $F_G(u) = F_{G'}(u)$, then

$$p(A_{ij}) = P(x'_k \in (x_{(i)}, x_{(j)})) = p_{ij} = \frac{j-i}{n+1}. \tag{1}$$

If we have a sample $x' \in (x'_{(1)}, x'_{(2)}, ..., x'_{(m)})$, we can find the frequency h_{ij} of the random events A_{ij} and the confidence intervals $(p_{ij}^{(1)}, p_{ij}^{(2)})$ for the probability p_{ij} at a given significance level β, i.e. $B = \{p_{ij} \in (p_{ij}^{(1)}, p_{ij}^{(2)})\}$, $p(B) = 1 - \beta$. Then

$$p_{ij}^{(1)} = \frac{h_{ij}m + \frac{g^2}{2} - g\sqrt{h_{ij}(1-h_{ij})m + \frac{g^2}{4}}}{m + g^2} \tag{2}$$

$$p_{ij}^{(2)} = \frac{h_{ij}m + \frac{g^2}{2} + g\sqrt{h_{ij}(1-h_{ij})m + \frac{g^2}{4}}}{m + g^2} \tag{3}$$

where g satisfies the condition $\varphi(g) = 1 - \beta/2$ ($\varphi(g)$ is the density of the normal distribution). Let's set $g = 3$. Let us denote by N all confidence intervals $I_{ij} = (p_{ij}^{(1)}, p_{ij}^{(2)})$ ($N = n(n-1)/2$) and L is the number of intervals I_{ij} that contain the probability p_{ij}, i.e. $p_{ij} \in I_{ij}$. Let $h = \rho(x, x') = L/N$.

The statistic h is called the p-statistic (Petunin statistic), it is a measure of the closeness $\rho(x, x')$ between the samples x and x'.

The p-statistic, which is an example of proximity measure based on confidence limits might be a useful loss function due to its robustness to outliers. Since it focuses on the central bulk of the data through confidence limits, it is less sensitive to extreme values that could otherwise dominate other loss functions. This makes it an excellent choice for tasks where robustness is critical, such

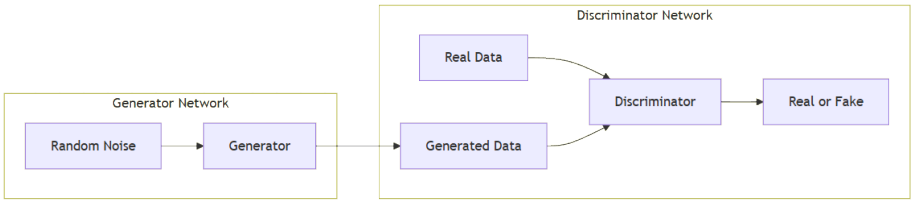

FIGURE 3. Generative Adversarial Network architecture.

as those involving noisy or imbalanced datasets. Furthermore, the proximity measure compares entire distributions rather than just individual data points or errors, which is especially valuable in tasks like generative modeling, where the goal is to match the distributions of generated and real data. Additionally, the use of confidence intervals means that this measure is well-suited for situations where uncertainty needs to be quantified, or where probabilistic predictions are essential.

However, there are considerations to keep in mind. First, the proximity measure could be computationally more complex because it involves order statistics and the calculation of confidence intervals. This makes it potentially more expensive in terms of computation compared to simpler loss functions like mean squared error or cross-entropy, so it may not be the best choice for large datasets or real-time applications. Another potential challenge is the interpretability of this loss function. Unlike more traditional loss functions, such as MSE or cross-entropy, the proximity measure might be harder to understand and explain. While it could offer advantages in specific domains, its use may require more effort to communicate and justify its application.

The p-statistic could be used as a loss function for training neural networks in scenarios where it is needed to compare the distributions of model outputs or predictions with some target distributions, or if it is needed to measure how well the neural network approximates a given distribution of data. One of potential scenarios in which p-statistic could be helpful as a neural network loss function is Generative Adversarial Networks distributions' matching.

A Generative Adversarial Network (GAN) is a type of machine learning model used for generative tasks, where the goal is to generate new, synthetic data that resembles a given set of real data. The architecture of a GAN consists of two neural networks: the generator and the discriminator (see Figure 3), which are trained in opposition to each other, hence the term «adversarial».

The generator network is responsible for producing fake data, such as images, from random noise or a latent space. The generator's aim is to create data that is indistinguishable from real data, based on the distribution it has learned from the training set.

The discriminator network, on the other hand, is trained to distinguish between real data (from the training set) and fake data (produced by the

generator). It outputs a probability score that represents the likelihood that the input data is real. Essentially, the discriminator's job is to correctly identify whether the data it receives is authentic or generated.

The two networks engage in a game-like process. The generator tries to «fool» the discriminator by generating increasingly realistic data, while the discriminator tries to get better at distinguishing real from fake data. This adversarial training process continues until the generator produces data that is so realistic that the discriminator can no longer reliably tell the difference between real and fake.

The training process of GANs typically involves optimizing the two networks in a way that the generator improves its ability to create more convincing fake data, and the discriminator improves its ability to correctly classify real versus fake data. This interplay between the generator and discriminator helps GANs generate high-quality, synthetic data that is often used in applications like image generation, text-to-image synthesis, super-resolution, and style transfer.

In this experiment, the primary goal is to investigate the effectiveness of using the p-statistic as a loss function in a Generative Adversarial Network. Typically, GANs are designed to generate data that resembles real-world samples, with the discriminator distinguishing between real and fake data, and the generator attempting to produce increasingly realistic data. Traditional loss functions like binary cross-entropy, Wasserstein distance, or Kullback-Leibler (KL) divergence are commonly used to compare the real and generated data at the sample level or the distribution level.

The p-statistic loss function, however, takes a different approach. Instead of directly comparing individual samples, the p-statistic utilizes confidence intervals and order statistics to evaluate how closely the generated data distribution aligns with the real data distribution. This method is expected to offer better robustness, especially in cases where the data contains noise or is imbalanced. This experiment evaluates whether the p-statistic loss can improve the generator's ability to produce realistic data when compared to more traditional loss functions.

The experiment aims to assess the effectiveness of the p-statistic-based loss in training a GAN, and to compare its performance against the more conventional loss functions. The key objectives are to determine whether the generator, when trained with the p-statistic loss, produces data that closely matches the real data distribution, and how the generator trained with the p-statistic performs in terms of image quality, distribution matching, and overall performance.

The GAN architecture used for this experiment consists of two main components: the generator and the discriminator. The generator takes random noise as input and aims to generate images that resemble real data, such as MNIST digits. The discriminator's task is to classify images as either real or fake, based on their similarity to real data.

Two different loss functions were used to train the GAN. The first was the p-statistic-based loss, which uses a proximity measure based on confidence intervals and order statistics to assess how closely the generated data matches

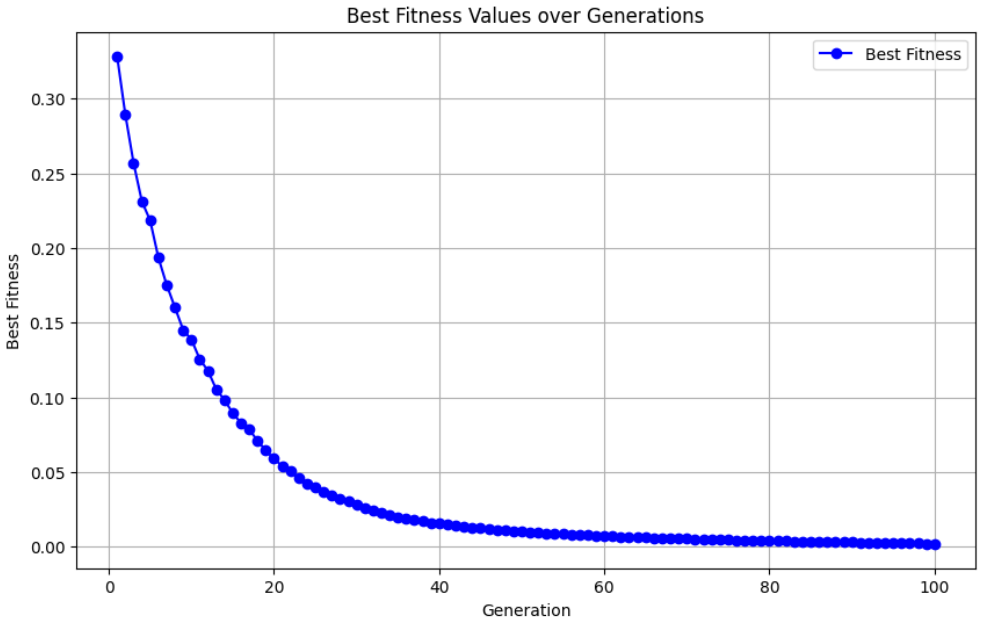

FIGURE 4. Learning curve of p-stat loss function.

the distribution of the real data. The second loss function was binary cross-entropy, a standard loss function used in GANs for training the generator and discriminator in adversarial settings.

Training was carried out for 100 epochs, with the goal of optimizing the generator to produce realistic data. The fitness score and visual inspection of the generated images were used to evaluate the quality of the generated samples.

During the course of the training, the p-statistic-based loss showed consistent convergence. This suggests that the p-statistic loss is less prone to overfitting early in training and focuses more on aligning the overall data distribution rather than focusing purely on individual samples (see Figure 4).

When inspecting the generated images, the results showed that the generator trained with the p-statistic loss produced diverse and realistic images, especially in the later stages of training. The p-statistic-based generator exhibited good generalization, producing varied images.

Additionally, the p-statistic-based loss demonstrated improved robustness to noisy or imbalanced data. When training on datasets with noisy labels or class imbalance, the generator trained with binary cross-entropy exhibited fluctuating performance, often failing to learn the underlying distribution effectively. On the other hand, the generator trained with the p-statistic-based loss maintained stable performance, producing high-quality images even in the presence of noisy or imbalanced data.

The experiment demonstrated that using the p-statistic-based loss for training a GAN resulted in solid distribution matching and high-quality generated

samples. The generator trained with the p-statistic loss showed good diversity in its generated samples and was able to closely mimic the distribution of real data.

One of the key strengths of the p-statistic loss is its ability to focus on the overall distributional similarity rather than on the pixel-level accuracy of individual samples. This characteristic makes the p-statistic loss particularly useful in scenarios where the goal is to generate data that aligns with the real distribution in a more global sense, rather than simply matching individual sample characteristics.

Overall, the results of this experiment highlight the potential of the p-statistic as an alternative loss function for GANs, offering better performance in certain contexts, especially when distributional similarity is paramount. Future work could focus on optimizing this approach to speed up convergence, as well as testing it on more complex datasets and real-world applications.

4. Conclusions

Optimizing neural networks with custom loss functions presents unique challenges that cannot always be addressed using conventional gradient-based methods. While backpropagation and its associated optimization algorithms, such as stochastic gradient descent and Adam, have been the cornerstone of deep learning, they rely on the assumption that the loss function is differentiable and can be efficiently computed. However, in many real-world applications, loss functions may be discontinuous, noisy, or defined by black-box processes, making traditional approaches ineffective. This limitation has led to the increasing interest in derivative-free optimization techniques, which do not require gradient information and can optimize models based solely on function evaluations.

Derivative-free optimization methods, including Bayesian optimization, evolutionary algorithms, and direct search techniques such as Nelder-Mead and Powell's method, offer alternative strategies for training neural networks when gradients are unavailable or unreliable. Evolutionary algorithms, such as genetic algorithms and covariance matrix adaptation evolution strategy, leverage population-based search strategies that enable global exploration and are particularly effective for highly non-convex and discontinuous objective functions. Bayesian optimization, on the other hand, provides a probabilistic framework for strategically selecting function evaluations, making it highly useful in cases where function evaluations are expensive and limited in number. Simpler direct search methods, like the Nelder-Mead simplex method and pattern search, offer efficient alternatives for low-dimensional problems but struggle to scale effectively in high-dimensional neural network training scenarios.

To illustrate the effectiveness of derivative-free optimization in real-world neural network training, we conducted an experiment where a Generative Adversarial Network was trained using a p-statistic-based custom loss function. Unlike conventional GAN training, which relies on differentiable losses such as binary cross-entropy or Wasserstein distance, our approach used a p-statistic metric to evaluate the similarity between generated and real samples. This custom loss

function was based on statistical hypothesis testing, making it non-differentiable and difficult to optimize using standard backpropagation.

This experiment highlights the practical applicability of derivative-free optimization in deep learning. By leveraging statistical metrics as custom loss functions, we can train models in scenarios where gradient-based approaches fail. However, the increased computational cost of derivative-free methods remains a key limitation, suggesting that future research should focus on developing more efficient hybrid optimization strategies that integrate gradient-based refinements with evolutionary or probabilistic search methods.

In conclusion, while gradient-based methods remain the dominant approach for optimizing neural networks, derivative-free optimization provides an essential toolset for handling complex, custom loss functions that challenge traditional techniques. By carefully selecting and adapting optimization methods to specific problem constraints, researchers and practitioners can expand the applicability of neural networks to a broader range of tasks, improving performance in scenarios where traditional optimization methods fall short. Moving forward, continued advancements in DFO methodologies, coupled with increasing computational power, will further enhance our ability to train neural networks effectively in diverse and complex environments.

The author declares that there is no conflict of interest concerning the publication of this article.

References

1. Kumar S. GD doesn't make the cut: Three ways that non-differentiability affects neural network training. *arXiv:2401.08426*. 2024.
2. Patel Y. Neural Network Training and Non-Differentiable Objective Functions. *arXiv:2305.02024*. 2023.
3. Audet C. Derivative-Free Optimization. *Springer Handbook of Computational Intelligence*. 2021.
4. Kim B., McKenzie D., Cai H., Yin W. Curvature-Aware Derivative-Free Optimization. *Journal of Scientific Computing* 2025. Vol. 103. P. 1–12.
5. Gross J. C., Parks G. T. Optimization by moving ridge functions: Derivative-free optimization for computationally intensive functions. *arXiv:2007.04893*. 2020.
6. Roberts L., Kirsch D., Hutter D. DFO-LS: Derivative-Free Optimizer for Least-Squares Minimization. *Numerical Algorithms Group* 2025.
7. Anggara D., Suarna N., Wijaya Y. Comparative analysis of Adam, SGD, and RMSprop optimizers performance on the H5 models. *Network Engineering Research Operation* 2023. Vol. 8. P. 53–64.
8. Huo Y. Convergence of Adam and RMSprop under Relaxed Smoothness Assumption. 2025.
9. Liu Y., Pan R., Zhang T. Large Batch Analysis for Adagrad Under Anisotropic Smoothness. *arXiv:2406.15244*. 2024.
10. Naaman D., Ahmed B., Ibrahim I. Optimization by Nature: A Review of Genetic Algorithm Techniques. *Indonesian Journal of Computer Science* 2025. Vol. 14. P. 1–14.
11. Chen D. Application of Improved Genetic Algorithms in Path Planning. 2024.

12. Uchida K., Yamaguchi T., Shirakawa S. Covariance Matrix Adaptation Evolution Strategy for Low Effective Dimensionality. *arXiv:2412.01156*. 2024.
13. Hansen N. The CMA Evolution Strategy: A Tutorial. *arXiv:1604.00772*. 2023.
14. Turner R., Eriksson D., McCourt M., Kiili J., Laaksonen E., Xu Z., Guyon I. Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge. *arXiv:2104.10201*. 2021.
15. Rio-Chanona E., Petsagkourakis P., Bradford E., Graciano J., Chachuat B. Real-Time Optimization Meets Bayesian Optimization and Derivative-Free Optimization: A Tale of Modifier Adaptation. *arXiv:2009.08819*. 2021.
16. Galantai A. A Stochastic Convergence Result for the Nelder–Mead Simplex Method. *Mathematics* 2023. Vol. 11. P. 1–10.
17. Selvam M., Manickam R., Saravanan V. Nelder–Mead Simplex Search Method – A Study. *Data Analytics and Artificial Intelligence* 2022. P. 117–122.
18. Ragonneau T. PDFO: a cross-platform package for Powell's derivative-free optimization solvers. *Mathematical Programming Computation* 2024. Vol. 16. P. 1–25.
19. Yavuz G. Senior Learning JAYA With Powell's Method and Incremental Population Strategy. *IEEE Access* 2022. Vol. 99. P. 1–19.
20. Klyushin D., Petunin Y. A Nonparametric Test for the Equivalence of Populations Based on a Measure of Proximity of Samples. *Ukrainian Mathematical Journal* 2003. Vol. 55. P. 147–163.

Received: 10.02.2025 / Accepted: 28.02.2025