STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **15**, June 2026, pp 4922–4943.
Published online in International Academic Press (www.IAPress.org)

Benchmarking Metaheuristics for Neural Network Optimization: A Comprehensive Study

Fakhita EDDAOUDI [1,*], Halima LAKHBAB [1], Mohamed NAIMI [2]

[1] *LAM2A Laboratory, Department of Mathematics and Computer Science, Hassan II University of Casablanca, Faculty of Science Aïn Chock, Morocco*
[2] *LAMSAD Laboratory, Department of Mathematics and Computer Science, Hassan First University, National School of Applied Sciences, Berrechid, Morocco*

Abstract
Deep learning is a machine learning method that has been successfully applied in various applications. However, these methods have structures and parameters that vary according to the specific application. Training a neural network is a highly complex task, but it is crucial to the process of learning the network. This step is generally carried out using gradient descent methods. Alternatively, metaheuristics are approximate optimization algorithms grounded in specific theories and objective functions that have shown better results in various domains. The idea of this work is therefore to use metaheuristics to train a deep neural network and compare the obtained results. To evaluate the different metaheuristics, we tested them on six different machine learning datasets. We have found that the PSO algorithm gives the best results in the training data. While its accuracy deteriorates on the testing data, we can conclude that the PSO is prone to overfitting. On the other hand, simulated annealing, and genetic algorithms generalize better to the test data. Comparing the execution times of the algorithms, we have found that simulated annealing is the fastest. This makes it the algorithm with the best trade-off between execution time and accuracy.

Keywords Metaheuristics, Neural Network Optimization, Deep Learning, Training

AMS 2010 subject classifications 68T05, 68Q25, 90C26, 90C59, 68W20.

DOI: 10.19139/soic-2310-5070-3060

1. Introduction

In 1943, Warren McCulloch and Walter Pitts [1] laid the foundations for neural networks by proposing the first mathematical model of an artificial neuron, inspired by the biological functioning of human neurons. A few years later, in 1958, Frank Rosenblatt [2] developed the Perceptron, a two-layer neural network capable of performing binary classification tasks and learning from labeled data. However, the Perceptron is a simple linear model using gradient descent to adjust its weights, it proved incapable of solving nonlinear problems, such as the famous XOR problem. At the same time, Rumelhart et al. [3] introduced the backpropagation (BP) algorithm. This algorithm enabled efficient training of Multi-Layer Perceptrons (MLPs), making it possible to solve nonlinear classification problems. Moreover, early research into gradient backpropagation (BP) revealed a risk of gradient explosion, posing a major challenge to their training.

Faced with these limitations, several approaches have been explored to optimize network performance. Geoffrey Hinton [4] made a significant contribution by proposing an innovative solution: he pre-trained the weights of Deep

*Correspondence to: Fakhita Eddaoudi (Email: fakhita.eddaoudi-etu@etu.univh2c.ma). Department of Mathematics and Computer Science, Hassan II University. Rue Tarik Ibnou Ziad, Casablanca (20360).

ISSN 2310-5070 (online) ISSN 2311-004X (print)

Neural Networks (DNNs) using unsupervised learning before refining these weights with supervised learning. This method proved to be a significant contribution to overcome the obstacles associated with training deep architectures.

At the same time, other researchers have explored the use of metaheuristics instead of gradient-based methods for the network training phase. For example, in [5], the authors compared the effectiveness of Particle Swarm Optimization (PSO) and Differential Evolution (DE) for training artificial neural networks. Similarly, work [6] highlights the application of Differential Evolution (DE), appreciated for its low computational complexity, to explore the weight search space for a non linear test applications. Paper [7] examines and compares various variants of Particle Swarm Optimization (PSO), an approach inspired by the collective intelligence of fish schools and bird flocks. Moreover, paper [8] presents the design, implementation, and experimental comparison of three relatively recent metaheuristics: Cuckoo Search, PSO, and Guaranteed Convergence Particle Swarm Optimization (GCPSO), applied to the Multi Layer Perceptron (MLP) training problem.

In our work, we will compare the performance of six metaheuristics in training a simple neural network for classification. We will start by exploring how metaheuristics can be used in training a neural network. In this case, the optimization problem will be formulated as a cost function minimization problem. We study six of the most used metaheuristics found in literature: the simulated annealing, the particle swarm optimization algorithm, the genetic algorithm, the tabu search algorithm, the ant colony optimization, and the bat algorithm. We benchmark these optimization algorithms on various classification datasets obtained from the UCI machine learning repository (https://archive.ics.uci.edu/). We will start this work by describing neural networks, as well as the metaheuristic algorithm we will use for their training. We then introduce the setup, formulating the training process of the neural network as an optimization problem. Then we describe the datasets used for the benchmark, and we present the empirical results of our study. Finally, we give some concluding remarks.

2. Multi-Layer Neural Network

2.1. Layers and Nodes

A Multi-Layer Perceptron (MLP) is typically organized into several distinct layers, each playing a unique role in information processing:

- **Input Layer:** This initial layer is responsible for receiving raw data into the system. The number of neurons in this layer directly corresponds to the count of features or attributes within the input dataset. Once received, this layer forwards the data to subsequent layers for further processing.

- **Hidden Layers:** These intermediate layers form the core of the network's processing capabilities. They act as bridges between the input and output layers, performing the majority of complex computations and nonlinear transformations necessary to identify patterns and relationships within the data. An MLP can incorporate one or multiple hidden layers, hence its "multi-layer" designation.

- **Output Layer:** The final layer of the network is dedicated to producing the model's result. The number of neurons in this layer depends on the specific problem being addressed (e.g., one neuron for binary classification, multiple for multi-class classification or regression).

The figure below shows an example of an MLP network (see Figure 1).

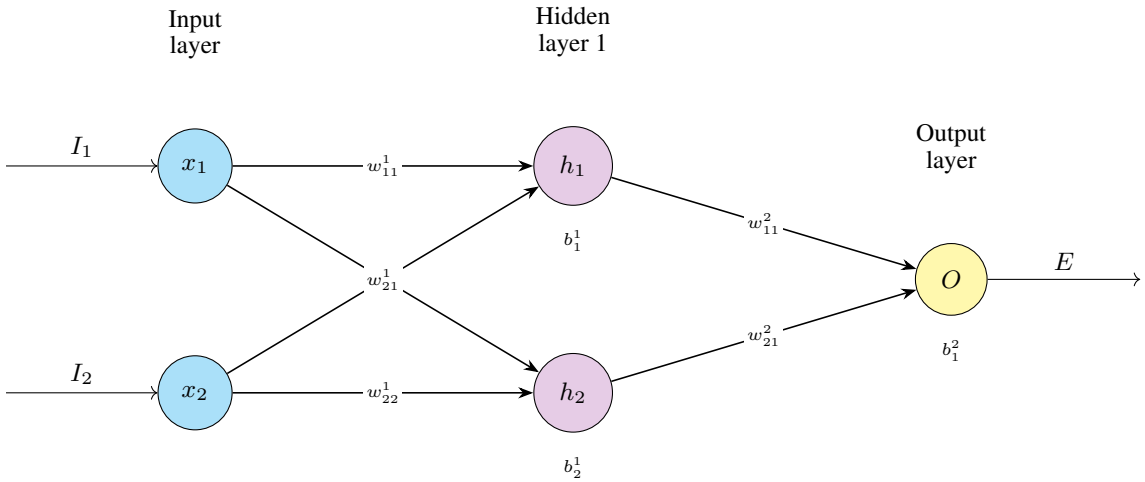


Figure 1. An MLP with two input neurons.

Each of these layers consists of a certain number of neurons. These neurons are interconnected, with information flowing between them via weighted connections. Specifically, the value transmitted from one neuron to another results from a linear combination of input values, multiplied by weights and often augmented by a bias, before being passed through a nonlinear activation function.

2.2. Activation function

In an artificial neural network, an activation function plays a crucial role in transforming a neuron's input signal into an output signal. This transformation typically follows a pattern that can be visualized as:

$$h_i(x) = f\left(W^T X + b\right) = f\left(\sum w_{ij} x_j + b_i\right) \tag{1}$$

Such as:

- W^T: is the transpose of the weight vector W of the neuron.
- h_i: is the output.
- X: is the input vector.
- b_i: is the bias.

For example, in Figure 1, we have a network featuring two inputs, x_1 and x_2. Each neuron in this setup possesses a distinct weight vector. Consequently, both inputs must be calculated with all the weighted vectors belonging to the neurons within the hidden layer.

$$h_1 = f\left(w_{11}^1 x_1 + w_{21}^1 x_2 + b_1^1\right) \tag{2}$$
$$h_2 = f\left(w_{21}^1 x_1 + w_{22}^1 x_2 + b_1^2\right) \tag{3}$$

Following this, the resulting outputs are then relayed to the output layer.

$$O = f\left(w_{11}^2 h_1 + w_{21}^2 h_2 + b_1^2\right) \tag{4}$$

The primary purpose of an activation function f within a neural network's input and hidden layers is to introduce non-linearity. This non-linearity is critical for the network to learn intricate patterns and relationships in data that

extend beyond basic linear correlations. Conversely, in a classification Deep Neural Network (DNN), the output layer's activation function is specifically designed to provide a standardized class probability for each category [9].

Among the most common activation functions are:

- Softmax: $f(h)_i = \frac{\exp(h_i)}{\sum_{k=1}^{K} \exp(h_k)}$
- Sigmoid function: $f(x) = \frac{1}{1+\exp(-x)}$
- Rectifier Linear Unit function: $f(x) = \max(0, x)$
- Linear function: $f(x) = \sum_i w_i x_i$

The selection of an activation function directly correlates with the problem at hand. For instance, in a multi-class classification problem, Deep Neural Networks (DNNs) are expected to output probabilities for each class; hence, a Softmax function is typically employed in the output layer. Conversely, for binary classification, a sigmoid function is often used. However, when tackling a regression problem, a linear activation function might be more suitable.

It's also crucial to recognize the significant benefit of using a Rectified Linear Unit (ReLU) function [10], or its variations, within the hidden layers of deep networks. These functions help mitigate the vanishing and exploding gradient problems, which can otherwise complicate the training of deep models.

2.3. Loss function

As is generally understood, most mathematical models inherently contain errors, often stemming from estimation factors. The concept of a loss function is used to quantify the total discrepancy between observed data and the model's output. Specifically, within a neural network, the loss function precisely defines the error between the expected (actual) results and the predicted (computed) results. The commonly used loss functions are [11]:

Table 1. Overview of loss functions and their mathematical formulation

Symbol	Classifier	Equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y} - \mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y} - \mathbf{o}\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \left\|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\right\|$
Hinge	hinge loss	$\sum_j \max\left(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)}\right)$
Hinge ^{2}	Squared hinge loss	$\sum_j \max\left(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)}\right)^2$
log	Cross entropy loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	Squared log loss	$-\sum_j \left[\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}\right]^2$
D_{CS}	Cauchy-Schwarz Divergence	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2 \|\mathbf{y}\|_2}$

2.4. Gradient descent

Training a neural network commonly occurs through the backpropagation algorithm. During this vital phase, the primary goal is to minimize the loss function to ensure strong performance for deep neural networks (DNNs). To achieve this, most machine learning methods rely on gradient descent techniques.

Since the optimization here aims to minimize a function (often treated as convex or locally convex for optimization purposes), calculating its gradient is essential. This involves determining the partial derivatives of the loss function concerning its adjustable parameters: the weights (w) and biases (b). Subsequently, we iteratively move in the opposite direction of this gradient, step by step, until a minimum (local or global) of the function is reached.

Before delving into the concept of gradient descent, it is essential to elucidate the notion of a function's derivative. The derivative represents the rate at which a function varies as we consider an infinitesimally small step

in the positive direction, providing a measure of its instantaneous change. Building upon this, the gradient emerges as a vector comprising partial derivatives, with each element reflecting the derivative concerning a specific variable upon which the function depends. In the context of a loss function, the gradient encapsulates partial derivatives with respect to each variable, such as the weights (w) and biases (b), facilitating a multidimensional analysis of the function's behavior. When endeavoring to minimize the function (c), defined on $v = (v_1, ..., v_n)$, it is instructive to initially conceptualize (c) as a function of two variables, to locate its minimum, constructing a graphical representation and identifying the lowest point is a viable approach. However, as (v) extends into $\mathbb{R}^n$, rendering a graph becomes impractical due to the increased dimensionality. An alternative strategy involves leveraging differential calculus to analytically determine the minimum, whereby computing derivatives and utilizing them to pinpoint extrema becomes a feasible methodology. Gradient descent is employed to iteratively determine the values of w_k and b_k for each iteration k that minimize the loss function. To elucidate its mechanism, the gradient vector's components are pivotal; expressing the update rule of gradient descent in terms of these components yields the iterative adjustments:

$$w_k = w_{k-1} - \alpha \frac{\partial c}{\partial w_{k-1}} \tag{5}$$

and

$$b_k = b_{k-1} - \alpha \frac{\partial c}{\partial b_{k-1}} \tag{6}$$

By repeatedly applying this update rule, it becomes possible to approximate the minimum of the loss function. This iterative process is instrumental in the training of neural networks, enabling the optimization of model parameters to enhance predictive accuracy and overall performance.

3. Metaheuristics

3.1. Simulated Annealing

In 1983, Scott Kirkpatrick et al. [12] introduced the concept of Simulated Annealing (SA) to the field of combinatorial optimization. This algorithmic approach draws direct inspiration from the physical annealing of solids. Imagine a metallurgical process where a material is heated to a high temperature and then cooled down very slowly. The goal is to allow its atoms to gradually rearrange themselves, settling into a state of minimal energy, which translates to a stable crystalline structure.

These physical principles are cleverly adapted into an optimization algorithm designed to tackle complex problems. The Simulated Annealing algorithm starts by randomly generating an initial solution. With each step, it explores a new neighboring solution by making a slight modification to the current one. Think of this little tweak as mirroring the random jiggling of atoms in a heated metal.

The solutions that improve the cost function are always accepted. However here's where it gets interesting: even if a move makes the cost function worse (increasing the energy), this less-than-ideal neighboring solution can still be chosen with a certain probability. This chance depends on two crucial factors:

- The algorithm's current temperature is a control parameter that slowly cools down over time.
- The energy degradation of the objective function ΔE. E simply represents the difference in the objective value between the current and the newly generated neighboring solution.

As the algorithm progresses, this temperature gradually drops. Consequently, the probability of accepting these bad moves also decreases. This innovative strategy allows the algorithm to broadly explore the solution space early on when the temperature is high. Then, as it cools, it begins to converge more precisely towards a minimum.

3.2. Tabu Search

The Tabu Search (TS) algorithm, introduced by Glover [16], functions similarly to a steepest local search method. However, a key distinction is its capacity to accept non-improving solutions when all neighboring solutions fail to offer improvement. This crucial feature enables TS to escape local optima.

Unlike SA, which often explores a random neighbor, TS typically conducts a deterministic exploration of the entire neighborhood. If a superior neighbor is identified, it replaces the current solution, much like in traditional local search. When the algorithm encounters a local optimum, the search continues by deliberately selecting a candidate that is inferior to the current solution. The best available neighbor is then chosen as the new current solution, even if it does not represent an improvement.

Tabu Search can be conceptualized as a dynamic modification of the search neighborhood. This strategy inherently risks revisiting previously explored solutions, potentially leading to cycles. To counteract this, TS employs a short-term memory, known as the tabu list, which records recently applied solutions or moves. At each iteration, this list is updated, preventing the algorithm from immediately reversing recent steps or revisiting specific configurations. Storing every visited solution would be computationally expensive; thus, the tabu list typically maintains a fixed number of tabu moves, often storing attributes of these moves rather than the complete solutions themselves.

While the tabu list effectively guides the search, it can, at times, be overly restrictive, potentially forbidding solutions that have not yet been generated or preventing the acceptance of a "good" move simply because it is tabu. To address this, TS incorporates aspiration criteria. These criteria define specific conditions under which a tabu-listed solution or move can nonetheless be accepted. Consequently, admissible neighbor solutions are those that are either not currently on the tabu list or satisfy these aspiration conditions.

3.3. Ant Colony Optimization

Ant Colony Optimization (ACO) algorithms [13] stand out as a prominent example within swarm intelligence methodologies. This computational approach simulates how ants behave when seeking and transporting food. The foundational version of this algorithm was introduced by Dorigo, conceptualized as a multi-agent system designed to tackle the Traveling Salesperson Problem.

The core concept behind ACO draws inspiration from the foraging habits of ant colonies. Initially, ants explore their surroundings randomly in search of food. Upon discovery, they carry a portion back to their nest. Critically, as they travel, they deposit pheromones along their path. The concentration of pheromone left on a trail is influenced by the quality and origin of the food source, and this chemical marker gradually evaporates. Residual pheromones on a given path can entice other ants to follow it. Consequently, over a short period, a majority of the ants tend to use the shortest available route, which is characterized by the highest pheromone concentration.

During the execution phase, ants generate various potential solutions haphazardly. These solutions are subsequently refined by adjusting the pheromone levels, a process tailored to the specific problem type and the traversal of the graph structure. Pheromones are typically distributed on either the nodes (vertices) or the edges of the graph. The likelihood of an ant choosing a specific path between two nodes is determined by a calculated probability, as follows:

$$p_{ij}^{k} = \frac{\tau(t)_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{j\in N^{k}(i)}\tau(t)_{ij}^{\alpha}\eta_{ij}^{\beta}} \tag{7}$$

In this context:

- p_{ij}^{k} represents the probability that an ant k will traverse the edge connecting node i to node j.
- $\tau(t)_{ij}^{\alpha}$ indicates the strength of the pheromone trail present on the edge between node i and node j, raised to the power of α.
- η_{ij}^{β} signifies the heuristic information associated with the edge from node i to node j, raised to the power of β.

- $N^k(i)$ refers to the collection of all nodes that ant k can potentially visit directly from its current position at node i.

To update the pheromone, we use the following equation:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{8}$$

3.4. Bat Algorithm

The Bat Algorithm was first developed by Yang in 2010 [14]. It draws inspiration from what we call echolocation, or the biosonar characteristics of microchiroptera (microbats).

Based on these echolocation features, Xin developed the Bat Algorithm, which follows these three main steps [15]:

- All bats use echolocation to sense the distance and location of their prey. They can also differentiate between their prey and obstacles.
- Bats fly randomly from a starting position x_0 with a velocity v_0, emitting a sound at a specific frequency f and an initial loudness A_0 to search for prey. They can adjust their pulse emission rate depending on their proximity to food.
- Loudness can vary in different ways; it's generally assumed that loudness decreases from an initial positive value A_0 down to a constant minimum value A_{Min}.

In the bat algorithm, each simulated bat at a given iteration t possesses a velocity v_i^t and a position x_i^t within a d-dimensional search space. Throughout the population, a current optimal solution, denoted as x^*, is identified. These core principles are mathematically represented by the following update equations for a bat's position and velocity:

$$\begin{aligned} f_i &= f_{\min} + (f_{\max} - f_{\min})\,\beta \\ v_i^t &= v_i^{t-1} + \left(x^* - x_i^{t-1}\right) f_i \\ x_i^t &= x_i^{t-1} + v_i^t \end{aligned} \tag{9}$$

Here, $\beta \in [0, 1]$ is a random vector sampled from a uniform distribution. With a probability r_i called pulse rate, the bat performs a local search by updating its position using the following equation:

$$x_i^t = x_i^{t-1} + rand\,\langle A^t \rangle \tag{10}$$

Where $rand$ is a random uniform number between $[-1, 1]$ and the $\langle A^t \rangle$ is the average loudness at each iteration t.

In the bat algorithm, we have to vary the loudness A_i and the pulse rate r_i to control the exploitation and exploration, and the switch between them. The update equations are as follows:

$$\begin{aligned} A_i^t &= \alpha A_i^{t-1} \\ r_i^t &= r_i^0 \left(1 - e^{-\gamma t}\right) \end{aligned} \tag{11}$$

Where $\alpha, \gamma \in [0, 1]$ and r_i^0 is the initial emission rate.

3.5. Particle Swarm Optimization

In 1995, Kennedy and Eberhart [17] proposed a new heuristic algorithm called Particle Swarm Optimization (PSO), inspired by the collective movement of bird flocks searching for food in space. In the PSO algorithm, each agent is considered a particle in a multidimensional space, defined by its velocity v_i and position x_i. The particles navigate the search space according to mathematical equations that update their positions and velocities. This update is based on the best performance achieved by the particle itself (its personal best) and the best position found by the

entire swarm (the global best). Equations 12 and 13 are used to update the position and velocity of each particle in every iteration.

$$v_i^{t+1} = wv_i^t + r_1c_1\,(p_i - x_i^t\,) + r_2c_2\,(g - x_i^t\,) \tag{12}$$

$$x_i^{t+1} = v_i^{t+1} + x_i^t \tag{13}$$

Where p_i is the personal best position found by the particle, while g is the global best position found by the swarm, the acceleration constants c_1 and c_2 are called the cognitive coefficient and the social coefficient respectively, and they modulate the magnitude of the steps taken by the particle in the direction of its personal best and global best, respectively. On the other hand, r_1 and r_2 are two random numbers generated from a uniform distribution in $[0, 1\,]$.

The velocity vectors in PSO dictate how particles navigate the search space. These vectors are a composite of three distinct influences:

- The inertia (or momentum) term acts as a dampener, preventing abrupt changes in a particle's movement by incorporating its prior direction of flow.
- The cognitive component reflects a particle's natural inclination to revert to its own most successful historical position (its personal best).
- The social component drives a particle's movement towards the best position discovered by the entire swarm, or, in the case of a local PSO variant, the best position within its immediate neighborhood.

3.6. Genetic Algorithm

The Genetic Algorithm (GA) stands as a widely recognized evolutionary algorithm, initially introduced by Holland in 1975 [18]. Its fundamental principle involves generating a population of potential solutions to a given problem, then allowing these solutions to evolve across successive generations to identify superior outcomes progressively.

Within the GA framework, a solution to an optimization problem is conceptualized as a chromosome. These chromosomes are comprised of genes, which represent the potential values of the design parameters (or decision variables) pertinent to the problem. The quality of these chromosomes (i.e., the solutions they embody) is quantitatively assessed by a fitness function, which evaluates the genetic information encoded within their genes.

Chromosomes within the GA are iteratively refined through the application of bio-inspired genetic operators: selection, crossover, and mutation. Chromosomes are subjected to a specified crossover probability p_c, and a mutation probability p_m. Should it be necessary, a repair operator can be employed to restore the feasibility of chromosomes that might violate problem constraints.

The selection and crossover operators are crucial in modulating the diversification and intensification strategies of the algorithm. Indeed, these two operators represent primary areas of focus for researchers aiming to enhance GA performance. Conversely, the mutation and repair operators play a more secondary, yet still important, role: they contribute to augmenting the genetic diversity within the population while ensuring chromosome feasibility when required.

4. Proposed method and experimental results

4.1. Proposed method

Feedforward neural networks are architectures in which data flows in a single direction, from the input layer through any hidden layers to the output layer, without any feedback or cycles. This absence of recurrent connections simplifies the way information is propagated through the network.

A crucial initial step in training such a network is the random initialization of weights and biases. However, this randomness does not guarantee optimal results, as poor initializations may trap the model in suboptimal local minima or hinder learning altogether.

The main objective of training is to make the network as accurate as possible in its predictions. In practical terms, this means that for each input, the output generated by the network should closely match the expected

output. This task is typically framed as a minimization problem of a cost function, usually denoted as $f(w, b)$, where w represents the network's weights and b the biases.

The cost function quantifies the network's prediction error across all training samples. Various loss functions can be used, as shown in Table 1. In our study, we focused on the cross-entropy function for classification tasks defined as:

$$f(w,b) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c}(y_{ij}\log(p_{ij})) \tag{14}$$

Where:

- n is the number of training samples.
- c is the number of classes.
- y_{ij} is 1 if class j is correct for sample i, and 0 otherwise.
- p_{ij} is the model-predicted probability of sample i being in class j.

This function is always positive. It reaches zero only when all the predicted outputs exactly match the actual labels. Therefore, a practical training algorithm finds w and b such that the cost function approaches zero.

However, in deep neural networks, gradient-based optimization methods face significant challenges. One major issue is the vanishing gradient problem, where gradients become very small as they propagate back through earlier layers, slowing or even halting the learning of those layers. Conversely, the exploding gradient problem occurs when gradients grow too large, causing instability. This gradient instability makes traditional learning approaches less effective for deep architectures.

To overcome these limitations, we implemented six different metaheuristic optimization techniques: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Tabu Search, the Bat Algorithm, Simulated Annealing, and Ant Colony Optimization (ACO). These methods do not rely on gradient information and instead explore the space of possible weights and biases globally. The goal is to find a set of weights and biases that minimizes the cost function, even in complex or non-differentiable search spaces.

In this context, each candidate solution (individual or particle) represents a vector of parameters, containing all weights and biases of the neural network. If the network has a total of D parameters, then each solution vector $x \in \mathbb{R}^D$ looks like:

$$x = [w_1, w_2, \ldots, w_p, b_1, \ldots, b_q] \tag{15}$$

The presented pseudocode (see Algorithm 1) illustrates the process of training a neural network using metaheuristic optimization techniques. The process begins with an initial set of candidate solutions, each defining a unique configuration for the network's parameters. For every solution, the network's output is generated and its error is calculated, which serves as a fitness metric. Solutions that produce a more minor error are considered more fit. Through an iterative process involving operators like selection and mutation, the metaheuristic algorithm refines these solutions. The cycle terminates when a stopping condition is fulfilled, and the most fit solution discovered is then implemented as the final parameter set for the trained network.

Algorithm 1 Training neural network using metaheuristics

Initialize a population of solutions (weights and biases) randomly: $\{w_i, b_i\}_{i=1}^{n}$
while Stopping criterion is not met **do**
  for each solution i in the population **do**
    Compute the network output with w_i, b_i
    Calculate the error function: E_i
    Assign fitness based on error: $f_i = E_i$
  end for
  Select, crossover, and mutate solutions according to the chosen metaheuristic (e.g., genetic algorithm, PSO, etc.)
end while
Select the best solution (w^*, b^*) obtained
Use (w^*, b^*) for final network

To adapt this approach to any given dataset, the following steps are taken:

- Match the input layer size to the number of features,
- Match the output layer size to the number of target classes,
- Generate candidate solutions of the appropriate size (total weights and biases),
- Evaluate each candidate using the cost function on the dataset.

Example using the Iris dataset:

- 4 input features $\rightarrow$ input layer of size 4.
- 3 classes $\rightarrow$ output layer of size 3
- Output encoding via one-hot vectors
- Use of softmax in the output layer and cross-entropy loss as the cost function

4.2. Dataset description

To assess the effectiveness of employing metaheuristics in the training of neural networks, we utilized six datasets for classification tasks. These datasets were sourced from the UCI Machine Learning Repository (https://archive.ics.uci.edu/). The characteristics of these datasets can be described as follows:

Credit Data: German credit data [19] is a widely used dataset in machine learning to evaluate classification algorithms. Its purpose is to assess credit risk by determining whether a loan applicant poses a low or high risk of default based on their financial and personal characteristics. The dataset includes 1000 instances, representing 1000 loan applicants, with approximately 70% considered likely to repay their loan (low risk) and 30% likely to default (high risk). It contains 20 attributes describing the applicant's financial status, employment, personal information, and credit history. For a neural network using this data for testing, there would be 20 inputs (corresponding to the attributes) and two outputs (representing the risk classes: low or high).

Iris Data: The Iris dataset, introduced by Ronald A. Fisher in 1936 [20], is a well-known resource for testing classification models. It comprises 150 samples of iris flowers, divided into three species: Iris setosa, Iris versicolor, and Iris virginica. Each sample is characterized by four features: sepal length, sepal width, petal length, and petal width, all measured in centimeters. When using this dataset for training a model, there would be four inputs (corresponding to the features) and three outputs (representing the species).

Prognostic Wisconsin Breast Cancer Data: The Wisconsin Prognostic Breast Cancer (WPBC) dataset [21], sourced from the University of Wisconsin, is a vital tool for predicting breast cancer recurrence in machine learning and medical research. Comprising 198 patient records, it includes 33 features: 30 derived from fine needle aspirate (FNA) images (mean, standard error, and worst values of 10 nuclear characteristics), plus tumor size, lymph node status, and an outcome variable (recurrent or non-recurrent with time-to-recurrence or disease-free survival data). Approximately 24% of cases are recurrent, and 76% are non-recurrent. Used for binary classification and survival analysis, it supports testing algorithms like neural networks, with 32 predictive inputs and two output classes. Despite minor missing data, its compact size and prognostic focus make it valuable for developing treatment-guiding models. Available under CC BY 4.0 at the UCI Machine Learning Repository, it is a key resource for cancer outcome prediction.

Wine Quality Data: The Wine Quality dataset [22], a widely adopted resource in machine learning for classification and regression tasks, compiles data on the physicochemical and sensory properties of red and white wines from the Vinho Verde region of Portugal. It is organized into two separate datasets: one for red wines, containing 1599 samples, and another for white wines, with 4898 samples. Each wine sample is characterized by 11 attributes, encompassing measures such as acidity, residual sugar, and alcohol content, alongside a quality score ranging from 0 to 10, assigned by expert evaluators.

Heart Disease Data: The Heart Disease dataset [23] facilitates the prediction of heart disease presence or absence in patients based on clinical and demographic data. It is primarily utilized for binary classification tasks, distinguishing patients with heart disease (positive class) from those without (negative class). Comprising 303 patient records, the dataset includes 13 input attributes encompassing demographic, clinical, and diagnostic test data, along with a target variable indicating either the absence (0) or presence $(1-4)$ of heart disease.

Glass Identification Data: The Glass Identification dataset [24] aims to categorize glass samples based on their chemical composition, serving forensic investigations by providing evidence to differentiate types of glass, such as window or container glass. Developed by B. German at the Central Research Establishment, Home Office Forensic Science Service (Aldermaston, United Kingdom), with contributions from Vina Spiehler, Ph.D., of Diagnostic Products Corporation, the dataset is derived from chemical analyses measuring the weight percentages of oxides in glass fragments. It encompasses 214 glass samples, each described by 10 attributes: 9 continuous features (e.g., refractive index, sodium, and calcium content) and one target variable specifying the glass type.

Table 2. Dataset specific Neural Network Architectures and references

DataSet	Number of Hidden Layers	Number of Neurons per Layer	activation function	Reference
Iris DataSet	3	3 for input layer. 4 for each hidden layer and 3 for the output layer	Hyperbolic tangent	F. Z. El-Hassani, M. Amri, N.E. Joudar, and K. Haddouch[25]
Credit DataSet	1	20 for the input layer, 10 for the hidden layer and 2 for the output Layer	Hyperbolic tangent	C. Guotai, M. Z. Abedin, and F. E.Moula [26]
Breast Cancer DataSet	1	33 for the input Layer, 100 for the hidden Layer and 2 for the output Layer	Rectified linear unit	M. H. Alshayeji, H. Ellethy, S. Abed, and R. Gupta [27]
Heart Disease DataSet	2	13 for the input Layer, 17 for the first hidden Layer and 16 for the second. 5 for the output Layer	Rectified linear unit	H. Bihri, R. Nejjari, S. Azzouzi, and M. E. H. Charaf [29]
Wine DataSet	1	11 for the input layer, 10 for the hidden layer and 7 for the output layer	Rectified linear unit	F. Z. El-Hassani, M. Amri, N.E. Joudar, and K. Haddouch[25]
Glass DataSet	3	9 for the input layer, 7 for the first hidden layer then 1 for the second and third hidden layers, 6 for the output Layer	Rectified linear unit	M. J. E.Khatib, B. S. A.Nasser, S. A.Naser [28]

4.3. Architecture of used Neural Networks

An important parameter that can influence the quality of the neural network model used for the classification tasks is the architecture of that neural network. As the goal of our study is to benchmark metaheuristics usage on optimisation models, we referred to the literature to find the best architecture to be used for these models. In the table 2, we list the architectures used for different datasets.

Table 3. Parameters used by metaheuristic classifiers for neural network training

Metaheuristics	Parameters	Value
Particle Swarm Optimization	Swarm size	30
	Inertia weight	0.9
	Cognitive	2
	Social	2
	Max velocity	0.5
	Min velocity	-0.5
Genetic Algorithms	Population size	50
	Crossover rate	0.8
	Mutation rate	0.1
	Elitism rate	0.1
	Tournament size	3
	Mutation size	0.1
Bat Algorithm	Number of bats	25
	Frequency min	0
	Frequency max	2
	Loudness initial	0.5
	Loudness reduction	0.95
	Pulse rate initial	0.5
	Pulse rate increase	1.05
Ant colony optimization	Number of ants	5
	Number of nodes	10
	Alpha	1
	Beta	2
	Evaporation rate	0.1
	Pheromone deposit	1
	Step size	0.1
Simulated Annealing	T0	100
	cooling rate	0.95
	Tmin	0.1
	iteration	100
Tabu Search	Tabu list size	100
	Tabu step	0.05

4.4. Parameter configuration

Metaheuristics are optimization strategies that rely on the meticulous tuning of their parameters to tackle complex problems effectively. Within the scope of our research, the selection of parameters for each metaheuristic, along with specific hyperparameters pertinent to neural networks, was conducted through a systematic tuning process to optimize performance. The following table summarizes these tuned parameters.

5. Results and Discussion

5.1. Results

To assess the effectiveness of metaheuristics in optimizing neural networks, a standard feedforward neural network was implemented, with each metaheuristic employed as a training mechanism. A series of experiments was designed to train the model using each metaheuristic, collecting results as the average fitness function value

throughout execution. The loss function utilized was cross-entropy. Each experiment was repeated 100 times based on the parameters outlined in Section 4.4. The experiments were implemented in $Python3.12.10$ and executed on the HPC MARWAN cluster `hpc.marwan.ma`. The performance of the metaheuristic-based classifiers was evaluated on six standard datasets, described in Section 4.2. The evaluated metrics include accuracy on the training and test sets (expressed through standard deviation, minimum, and maximum values), execution time (minimum and maximum, in seconds), the average Cross Entropy error and the generalization Gap, calculated using the following equation:

$$Gap = \text{Mean Training accuracy} - \text{Mean Testing accuracy} \tag{16}$$

The following tables present the results for each dataset.

Table 4. Statistical summary of training accuracy, testing accuracy, execution time, and average error for different metaheuristic classifiers and the SGD baseline applied to the Iris dataset

	Training accuracy			Testing accuracy			Execution time			Average Error	Gap
Classifier	Std	Min	Max	Std	Min	Max	Std	Min	Max		
ACOClassifier	9.82	35.24	92.38	13.80	28.89	88.89	1.14	7.10	15.97	0.88	-2.2
BatClassifier	13.93	22.86	89.52	18.11	17.78	93.33	4.55	32.80	57.22	0.86	-1.3
GAClassifier	12.43	64.76	97.14	11.44	55.56	100.00	6.27	52.49	90.86	0.60	-2.3
PSOClassifier	12.60	60.00	97.14	11.74	42.22	100.00	2.85	29.50	41.99	0.59	-1.2
SAClassifier	14.61	29.52	84.76	19.57	17.78	84.44	0.38	1.63	3.11	1.02	4.4
TabuClassifier	13.24	64.76	97.14	11.70	71.11	100.00	0.89	41.46	46.04	0.55	-3.2
SGD	6.49	29.52	62.85	6.56	28.89	71.11	0.8	6.6	12.4		3.72

Table 4 provides a statistical summary of the performance of different metaheuristic classifiers alongside the classical Stochastic Gradient Descent (SGD) baseline when applied to the Iris dataset. The ACOClassifier exhibits the lowest performance, with Training accuracy ranging from 35.24% to 92.38% with a standard deviation 9.82, and Testing accuracy between 28.89% and 88.89%, with a standard deviation 13.80, accompanied by an average error of 0.88. Both GAClassifier and PSOClassifier achieve high accuracy, with Training accuracy between $64.76\% - 97.14\%$ and $60\% - 97.14\%$, respectively, and Testing accuracy reaching 100% in both cases, with a standard deviations of 11.44 and 11.74. TabuClassifier also attains a maximum Testing accuracy of 100%, with slightly higher minimum test accuracy 71.11% and a low average error 0.55. SAClassifier demonstrates the shortest execution times, ranging from 1.63 to 3.11 seconds, and standard deviation 0.38, whereas other classifiers, such as BatClassifier and GAClassifier, require longer execution times (up to 57.22 and 90.86 seconds, respectively). Overall, GA, PSO, and TabuClassifiers provide the most accurate predictions with relatively low errors, while SAClassifier offers the fastest computation. In significant contrast, the gradient-based baseline (SGD) underperformed in this specific configuration, with a maximum testing accuracy of only 71% and a maximum training accuracy of 62.85%. Generalization tends to be interesting when measured by the Gap metric. The negative gaps of the majority of metaheuristics (ACOClassifier, BatClassifier, GAClassifier, PSOClassifier, TabuClassifier) were negative. This effect of high test accuracy compared to training accuracy on Iris datasets, indicates that the models have overgeneralized to the test split, which is not seen in the training set, and is not overfitting. SAClassifier, on the other hand, exhibited a positive gap of 4.4, which shows that it slightly tends overfitting as in the other metaheuristics.

Table 5. Statistical summary of training accuracy, testing accuracy, execution time, and average error for different metaheuristic classifiers applied to the breast Cancer dataset

	Training accuracy			Testing accuracy			Execution time			Average Error	Gap
Classifier	Std	Min	Max	Std	Min	Max	Std	Min	Max		
ACOClassifier	4.30	55.80	78.99	7.05	46.67	80	0.89	11.85	16.35	2.10	2.7
BatClassifier	2.22	73.19	84.78	3.28	66.67	85	3.50	38.39	61.51	0.48	0.7
GAClassifier	2.25	81.88	92.75	3.40	68.33	86.67	5.46	43.85	83.42	0.31	8.1
PSOClassifier	0.00	100	100	3.62	68.33	85	2.03	31.90	40.88	0.02	23.7
SAClassifier	2.03	74.64	86.23	2.50	73.33	85	0.42	1.26	4.56	0.44	0.5
TabuClassifier	0.07	99.28	100	4.26	63.33	86.67	1.81	47.52	57.32	0.01	25.04
SGD	0	75.36	75.36	0	78.3	78.3	3.5	25.2	40		-2.97

Table 5 summarizes the performance of various metaheuristic classifiers and the SGD baseline on the Breast Cancer dataset. ACOClassifier shows the lowest accuracy (training 55.80–78.99%, testing 46.67–80%) with a high average error (2.10). BatClassifier and GAClassifier improve performance, reaching testing accuracies up to 85% and 86.67% with moderate errors (0.48 and 0.31). PSOClassifier and TabuClassifier achieve near-perfect training accuracy (100% and 99.28–100%) and strong testing results, with minimal errors (0.02 and 0.01). SAClassifier is the fastest, requiring only 1.26–4.56 seconds. Overall, PSOClassifier and TabuClassifier provide the most accurate predictions, while SAClassidfier offers efficient computation.

The SGD baseline showed extreme stability (Std = 0), likely converging to the same local optimum in every run. However, it stagnated at a testing accuracy of 78.3%, which is significantly lower than the best metaheuristics (GAClassifier and TabuClassifier reached 86.67%). This confirms that for this specific non-convex problem landscape, metaheuristic global search mechanisms offer a tangible advantage over simple gradient descent.

Table 6. Statistical summary of training accuracy, testing accuracy, execution time, and average error for different metaheuristic classifiers applied to the Credit dataset

	Training accuracy			Testing accuracy			Execution time			Average error	Gap
Classifier	Std	Min	Max	Std	Min	Max	Std	Min	Max		
ACOClassifier	1.60	65.43	72.71	2.82	62	76	1	11.20	15.48	0.61	0.61
BatClassifier	0.76	69	73	1.31	64	74	2.93	44.17	63.34	0.57	0.56
GAClassifier	0.92	74.43	78.29	1.14	72.33	78	5.08	51.93	85.29	0.48	1.24
PSOClassifier	1.24	83.71	90.14	2.09	67	77	2.10	37.21	49.43	0.32	14.5
SAClassifier	1.35	69.57	75.43	1.73	69	78.33	0.23	1.42	2.29	0.54	- 0.01
TabuClassifier	1.12	83.14	88.14	1.86	66.33	75	1.94	49.75	61.83	0.33	15.58
SGD	0	70.14	70.14	0	69.7	69.7	5	47.7	71.3		0.47

Table 6 summarizes the performance of various metaheuristic classifiers on the Credit dataset. ACOClassifier demonstrates the lowest accuracy, with training accuracy ranging from 65.43% to 72.71% and testing accuracy between 62% and 76%, accompanied by an average error of 0.61. BatClassifier and GAClassifier show moderate improvements, achieving testing accuracies up to 74% and 78% with average errors of 0.57 and 0.48, respectively.

PSOClassifier and TabuClassifier exhibit the highest predictive performance, with PSOClassifier reaching training accuracy of 83.71–90.14% and testing accuracy of 67–77%, and a low average error of 0.32. TabuClassifier shows comparable results, with training accuracy up to 88.14% and testing accuracy up to 75%, maintaining a low average error (0.33). However, this aggressive optimization led to significant overfitting, as evidenced by the large positive generalization gaps of 14.5% (PSOClassifier) and 15.58% (TabuClassifier). SAClassifier achieves moderate accuracy but stands out for its computational efficiency, requiring only 1.42–2.29 seconds. Overall, PSO and Tabu classifiers provide a favorable balance between accuracy and error, while SAClassifier offers the

fastest execution. The SGD baseline struggled on this dataset, achieving a maximum testing accuracy of only 69.7%, which is nearly 8.6% lower than the best metaheuristic result (SAClassifier). Furthermore, SGD required significant computational time (Max 71.3s) to converge. This result strongly validates the use of metaheuristics like SA , which not only outperformed the gradient-based baseline in accuracy but did so with drastically reduced execution time (Max 2.29s for SAClassifier).

Table 7. Statistical summary of training accuracy, testing accuracy, execution time, and average error for different metaheuristic classifiers applied to the Glass dataset

	Training accuracy			Testing accuracy			Execution time			Average error	Gap
Classifier	Std	Min	Max	Std	Min	Max	Std	Min	Max		
ACOClassifier	2.33	33.56	44.97	3.49	26.15	43.08	0.58	8.85	10.71	1.50	2.49
BatClassifier	0.73	32.89	36.91	3.05	29.23	35.38	2.33	36.37	52.39	1.56	2.33
GAClassifier	7.04	35.57	59.06	5.72	30.77	58.46	3.26	41.62	65.96	1.40	1.98
PSOClassifier	8.53	34.23	68.46	7.29	29.23	66.15	1.74	29.59	36.04	1.35	1.22
SAClassifier	5.17	12.75	47.65	4.93	15.38	40.00	0.09	1.17	1.52	1.59	1.94
TabuClassifier	8.30	34.23	63.09	7.39	29.23	63.08	2.81	39.40	55.34	1.35	1.6
SGD	1.11 e^{-14}	34.22	34.22	1.12e^{-16}	29.23	29.23	2.25	22.3	34.2		4.99

Table 7 presents a statistical summary of the performance of different metaheuristic classifiers on the Glass dataset. Overall, predictive performance is lower compared to previous datasets. ACOClassifier and BatClassifier show relatively modest training accuracies (33.56–44.97% and 32.89–36.91%) and testing accuracies (26.15–43.08% and 29.23–35.38%), with average errors of 1.50 and 1.56, respectively.

GAClassifier, PSOClassifier, and TabuClassifier achieve higher training and testing accuracies, reaching up to 59.06%, 68.46%, and 63.09% in training, and 58.46%, 66.15%, and 63.08% in testing, with lower average errors (1.40–1.35). SAClassifier maintains comparable accuracy but stands out for its computational efficiency, with execution times as low as 1.17–1.52 seconds. These results indicate that PSO and Tabu classifiers provide a favorable balance between accuracy and error, while SAClassifier is the fastest, albeit with slightly lower predictive performance. A critical finding is the stagnation of the SGD optimizer. With a maximum Testing accuracy of only 29.23%, SGD failed to converge to a useful solution. In stark contrast, PSO improved upon this baseline by nearly 37 percentage points. This result strongly supports the hypothesis that metaheuristics possess superior global search capabilities necessary for navigating complex, multi-modal loss landscapes where gradient descent becomes trapped in poor local optima.

Table 8. Statistical summary of training accuracy, testing accuracy, execution time, and average error for different metaheuristic classifiers applied to the Heart Disease dataset

	Training accuracy			Testing accuracy			Execution time			Average error	Gap
Classifier	Std	Min	Max	Std	Min	Max	Std	Min	Max		
ACOClassifier	3.52	41.04	58.96	4.24	37.36	57.14	0.42	9.43	11.23	1.32	3.29
BatClassifier	0.43	54.25	58.49	0.36	51.65	56.04	3.08	34.90	52.68	1.32	1.99
GAClassifier	2.00	63.68	73.11	2.78	48.35	61.54	2.20	38.76	52.46	0.77	12.5
PSOClassifier	2.41	75.00	87.74	3.64	43.96	62.64	1.20	29.31	34.00	0.52	27.5
SAClassifier	0.00	54.72	54.72	0.00	52.75	52.75	0.26	1.12	2.35	1.28	1.96
TabuClassifier	2.16	75.47	85.38	3.78	43.96	61.54	1.85	40.93	50.08	0.50	28.12
SGD	0	54.7	54.7	0	52.7	52.7	3.4	27.7	43	0.50	1.97

Table 8 summarizes the performance of various metaheuristic classifiers on the Heart Disease dataset. PSOClassifier and TabuClassifier achieve the highest training accuracies (75.00–87.74% and 75.47–85.38%)

and moderate testing accuracies (43.96–62.64% and 43.96–61.54%) with low average errors (0.52 and 0.50). GAClassifier attains moderate accuracy, while ACOClassifier and BatClassifier show the lowest predictive performance. SAClassifier demonstrates stable but limited accuracy, with very fast execution times. Overall, PSOClassifiers and TabuClassifiers provide a good balance between predictive performance and computational efficiency.

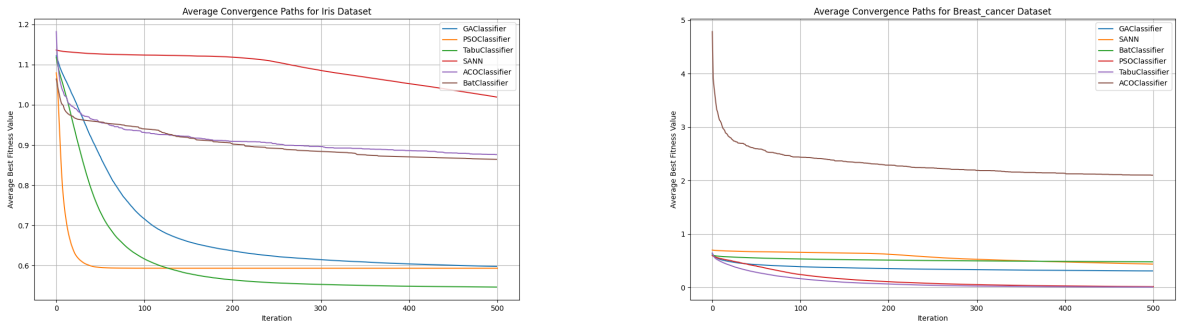
The PSOClassifier achieved the highest maximum training accuracy of 87.74%, significantly outperforming the baseline SGD. However, this aggressive exploration led to severe overfitting, as indicated by the substantial generalization gap of 27.5% and a drop in testing accuracy to 62.64%. Similarly, TabuClassifier Search exhibited a gap of 28.12%.

Table 9. Statistical summary of training accuracy, testing accuracy, execution time, and average error for different metaheuristic classifiers applied to the Wine dataset

	Training accuracy			Testing accuracy			Execution time			Average error	Gap
Classifier	Std	Min	Max	Std	Min	Max	Std	Min	Max		
ACOClassifier	4.57	25.20	46.25	5.12	25.08	48.87	1.00	14.97	18.36	1.84	-0.57
BatClassifier	3.45	30.68	50.25	4.45	29.28	49.44	8.59	70.28	134.07	1.57	-1.69
GAClassifier	0.61	52.76	55.66	0.71	51.23	54.67	5.00	95.55	126.27	1.11	1.22
PSOClassifier	0.55	54.85	57.33	0.69	52.72	55.85	2.24	63.17	72.82	1.04	1.94
SAClassifier	2.27	34.92	48.43	2.90	32.77	49.18	266.61	1.94	1042.13	1.49	-2.24
TabuClassifier	0.59	53.62	56.63	0.82	51.79	55.64	1.52	61.88	71.87	1.06	1.58
SGD	0.2	53.8	55.8	0.3	52.5	54.3	42.8	292.2	520.3		1.58

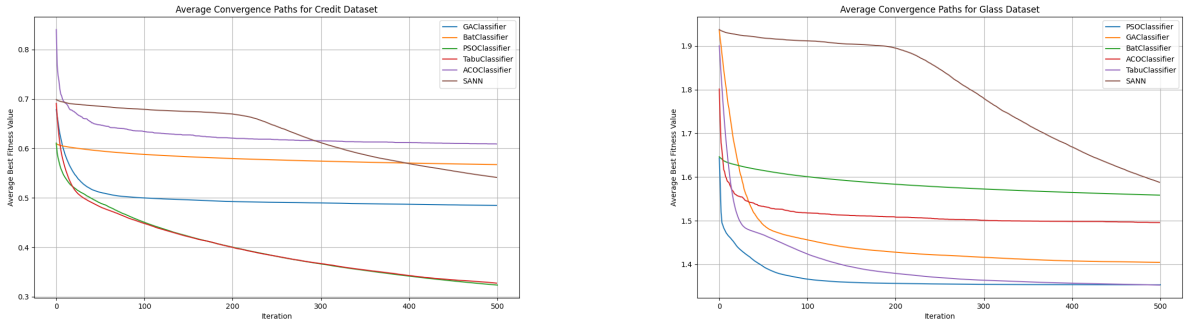
Table 9 presents the performance of metaheuristic classifiers on the Wine dataset. PSOClassifier and TabuClassifier achieve the highest and most consistent training and testing accuracies (54.85–57.33% and 52.72–55.85% for PSO; 53.62–56.63% and 51.79–55.64% for Tabu), with relatively low average errors (1.04 and 1.06). GAClassifier shows moderate accuracy and error (1.11), while ACOClassifier and BatClassifier perform the worst, with lower accuracies and higher errors (1.84 and 1.57). SAClassifier exhibits highly variable execution times, ranging from 1.94 to 1042.13 seconds, despite moderate accuracy (1.49 average error). Overall, PSO and Tabu classifiers provide the best trade-off between predictive performance and stability.

To complement the quantitative analysis provided in the comparative tables and to offer a more visual and dynamic perspective, the following figures illustrate the convergence behavior of the studied algorithms. Each graph depicts the progression of the mean error function as a function of the number of epochs.



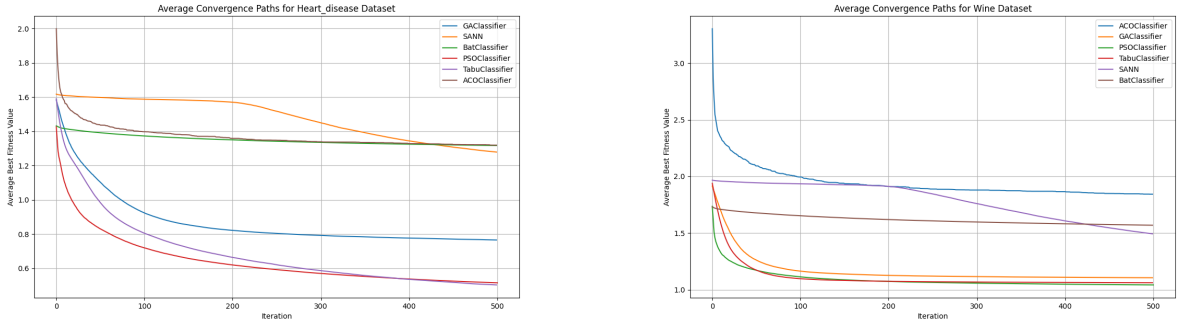
(a) Iris (b) Cancer

Figure 2. Evolution of the average error function value as a function of the number of epochs for each metaheuristic across Iris and Cancer datasets.



(a) Credit (b) Glass

Figure 3. Evolution of the average error function value as a function of the number of epochs for each metaheuristic across credit and Glass datasets.



(a) Heart (b) Wine

Figure 4. Evolution of the average error function value as a function of the number of epochs for each metaheuristic across Heart disease and Wine datasets.

The plots illustrate the average convergence paths of various classifiers across six distinct datasets: Breast Cancer, Credit, Heart Disease, Iris, Glass, and Wine. Each plot displays the average best fitness value on the y-axis against the number of iterations on the x-axis, with a range of 0 to 500 iterations. The classifiers evaluated include GAClassifier, SAClassifier , BatClassifier, PSOClassifier, TabuClassifier, and ACOClassifier, each represented by distinct colored lines.

For the Breast Cancer datasets as shown in Figure2b, the ACOClassifier exhibits the highest initial average best fitness value, starting near 5 and decreasing sharply to stabilize around 2 after approximately 100 iterations. In contrast, other classifiers such as GAClassifier, SAClassifier , BatClassifier, PSOClassifier, and TabuClassifier converge more gradually, stabilizing at values below 1 after 200 iterations, indicating a slower but more consistent convergence.

For the Iris datasets in Figure 2a, the initial average best fitness values are lower, with ACOClassifier peaking near 1.2 and declining, while PSOClassifier, GAClassifier, BatClassifier, and TabuClassifier converge to values between 0.6 and 1.0, with SAClassifier maintaining a higher stable value around 1.1.

The Credit datasets plot in Figure 3a shows a similar trend, with ACOClassifier beginning at the highest average best fitness value (around 0.8) and declining steeply, while GAClassifier, SAClassifier , BatClassifier,

PSOClassifier, and TabuClassifier converge to values between 0.5 and 0.6, with minimal variation after 200 iterations.

The Glass datasets plot in Figure 3b indicates that ACOClassifier starts at the highest value (near 1.9) and decreases, while other classifiers converge to values between 1.4 and 1.6, with PSOClassifier and GAClassifier showing the lowest convergence points around 1.4.

In the Heart Disease datasets as observed in Figure 4a, ACOClassifier again starts with the highest value (approximately 2.0) and decreases significantly, while the other classifiers converge to values ranging from 0.6 to 1.0, with BatClassifier showing a relatively stable path around 1.4.

Finally, the Wine dataset 4b shows ACOClassifier with the highest initial value (around 3.0) and a sharp decline, while PSOClassifier, GAClassifier, BatClassifier, and TabuClassifier stabilize between 1.0 and 1.5, with SAClassifier converging to the lowest value around 1.5.

Overall, the plots suggest that ACOClassifier generally exhibits the highest initial average best fitness values across all datasets, followed by a rapid convergence. Other classifiers demonstrate more gradual convergence, stabilizing at lower fitness values, which may indicate differing efficiencies or optimization strategies tailored to the specific datasets.

Table 10. T-test comparison of different classifiers

	P value					
Comparison	Breast cancer	Credit	Iris	Wine	Heart disease	Glass
SAClassifier vs GAClassifier	2.4e-3	1.05e-36	1.2e-41	2.7e-67	2.8e-16	2.5e-14
SAClassifier vs ACOClassifier	4.4e-39	6.6e-22	2.1e-8	3.1e-29	8.3e-11	6.8e-2
SAClassifier vs TabuClassifier	5.6e-19	4.4e-13	4.6e-45	1.3e-72	1.6e-1	6.7e-18
SAClassifier vs PSOClassifier	2.7e-14	9e-1	3.6e-37	4.6e-78	3.3e-1	5.5e-20
SAClassifier vs BatClassifier	3.5e-9	7.3e-15	4.0e-6	9.7e-6	3.7e-1	4.4e-1
GAClassifier vs ACOClassifier	9.1e-32	1.7e-56	6.9e-32	9.2e-77	3.3e-21	1.0e-12
GAClassifier vs TabuClassifier	4.3e-10	2.8e-60	8.4e-2	2.6e-10	8.6e-10	1.8e-2
GAClassifier vs PSOClassifier	5.8e-6	1.0e-30	5.9e-2	2.2e-29	5.8e-6	3.0e-3
GAClassifier vs BatClassifier	8.8e-3	1.0e-73	1.0e-26	8.5e-58	9.8e-16	3.7e-16
ACOClassifier vs TabuClassifier	1.2e-16	3.0e-6	2.6e-36	9.8e-80	4.4e-5	2.1e-16
ACOClassifier vs PSOClassifier	2.5e-22	9.2e-20	3.6e-26	1.4e-82	2.1e-8	1.2e-18
ACOClassifier vs BatClassifier	3.9e-27	4.5e-8	5.6e-1	1.2e-13	6.2e-11	1.5e-1
TabuClassifier vs PSOClassifier	2.5e-2	6.0e-11	4.1e-4	6.8e-8	9.1e-2	5.9e-1
TabuClassifier vs BatClassifier	2.3e-5	5.3e-1	1.6e-30	8.0e-62	1.3e-1	2.4e-19
PSOClassifier vs BatClassifier	3.0e-2	8.1e-12	4.4e-22	1.4e-65	3.8e-1	9.3e-22

To statistically validate the performance differences among various metaheuristic classifiers and the SGD baseline across six distinct datasets, a t-test analysis was conducted (see Table 10). This parametric test assesses whether the observed disparities in accuracy between pairs of classifiers are attributable to random variation or reflect genuine differences in algorithmic efficacy. The p-values, reported in the table, quantify the probability of observing the obtained results, or more extreme ones, assuming that the null hypothesis is true , which posits that no significant difference exists between the compared classifiers. A p-value below a conventional threshold (e.g., 0.05) indicates statistical significance, suggesting that the performance divergence is unlikely to occur by chance.

For instance, the comparison between GAClassifier and ACOClassifier yields notably low p-values across all datasets (e.g., $9.1e-32$ for Breast Cancer, $1.7e-56$ for Credit), underscoring a substantial difference in performance. Conversely, higher p-values, such as 0.9 for PSOClassifier vs. SAClassifier on the Credit dataset, imply that their performance differences may not be statistically significant, indicating comparable efficacy in certain contexts. The analysis further reveals dataset-specific patterns, with p-values varying widely (e.g., $2.4e-3$ for SAClassifier vs. GAClassifier on Breast Cancer vs. $1.2e-41$ on Iris), highlighting the influence of data characteristics on classifier robustness.

Table 11. Statistical analysis using the Friedman Test.

Test	Statistic	P-value
Friedman	23.214	< 0.001

Unlike the T-test, which is limited to pairwise comparisons, the Friedman test assesses whether there are statistically significant differences in the ranking of algorithms across multiple datasets. The results of the Friedman test are presented in Table 11. The calculated test statistic is 23.21 with a corresponding p-value of 0.00073. Since this p-value is well below the significance threshold ($\alpha = 0.05$), we reject the null hypothesis, concluding that there are significant differences in the performance distributions of the tested algorithms.

Table 12. Nemenyi Post-Hoc Test P-Values: Pairwise Comparison

Method	SA	GA	ACO	Tabu	PSO	Bat	SGD
SA	1.000	0.763	0.893	1.000	0.893	1.000	0.145
GA	0.763	1.000	0.105	0.937	1.000	0.591	**0.001**
ACO	0.893	0.105	1.000	0.680	0.196	0.967	0.835
Tabu	1.000	0.937	0.680	1.000	0.985	0.994	0.051
PSO	0.893	1.000	0.196	0.985	1.000	0.763	**0.003**
Bat	1.000	0.591	0.967	0.994	0.763	1.000	0.258
SGD	0.145	**0.001**	0.835	0.051	**0.003**	0.258	1.000

Following the rejection of the null hypothesis, we conducted the Nemenyi post-hoc test to identify exactly which pairs of algorithms differ significantly. The pairwise p-values are summarized in Table 12. The analysis reveals two critical insights. First, significant statistical differences were observed between the population-based metaheuristics and the gradient-based baseline. Specifically, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) demonstrated statistically significant superiority over Backpropagation, with p-values of 0.001 and 0.003, respectively. Tabu Search also showed a strong difference compared to Backpropagation, with a borderline p-value of 0.051. Second, the comparisons among the metaheuristics themselves (e.g., SA vs. GA, PSO vs. Tabu) yielded high p-values (typically $p > 0.5$ or close to 1.0). This indicates that while their raw accuracies may vary on specific datasets, there is no statistically significant difference in their overall rankings across the test suite. This suggests that the metaheuristic classifiers form a statistically distinct and superior group compared to the standard Backpropagation baseline for these specific architectures.

This statistical approach enhances the reliability of the benchmarking study by providing a quantitative basis for comparing metaheuristic optimization techniques, complementing the descriptive metrics of accuracy and execution time. Such insights are instrumental for guiding future refinements, enabling researchers to prioritize algorithms with proven superiority and tailor their application to specific problem domains based on empirical evidence of statistical distinction.

5.2. Discussion

This work, integrating a standard feedforward neural network optimized through various metaheuristic algorithms, provides a comprehensive framework for evaluating the efficiency of optimization techniques in neural network training. The experiments were conducted using Python 3.12.10 on the HPC MARWAN cluster `hpc.marwan.ma` and repeated 100 times with parameters specified in Section 4.4, covering six datasets: Iris, Breast Cancer, Credit, Glass Identification, Heart Disease, and Wine Quality. Cross-entropy was used as the loss function, and performance was assessed using training and testing accuracy (with standard deviation, Minimum, and maximum values), execution time, and average error.

The results, summarized in Tables 4, 5, 6, 7, 8, and 9, reveal clear performance trends among the metaheuristic classifiers and the gradient-based baseline.

PSOClassifier and TabuClassifier consistently achieve the highest training accuracies, particularly on the Breast Cancer dataset (up to 100%) with minimal average errors (0.02 and 0.01, respectively). However, an analysis of the generalization Gap reveals a significant tendency toward overfitting in these algorithms. For instance, on the Heart Disease and Breast Cancer datasets, both classifiers exhibited large gaps exceeding 20%, indicating that while they aggressively minimize training error, they struggle to generalize to unseen data.

In contrast, SAClassifier emerges as a robust alternative. It not only excels in computational efficiency, showing the shortest execution times across most datasets (e.g., 0.26 — 2.35 seconds on Heart Disease), but also demonstrates superior generalization capabilities with minimal gaps (e.g., $< 1\%$ on Credit). Furthermore, comparisons with the SGD baseline highlight the distinct advantage of metaheuristics on non-convex landscapes. While SGD provided a stable baseline, it frequently underperformed in terms of peak accuracy, most notably on the Glass dataset where it stagnated at 29.23% compared to PSO's 66.15%. This confirms that metaheuristics offer a tangible benefit in escaping local optima where standard gradient descent methods may fail.

Finally, ACOClassifier consistently underperforms, exhibiting lower accuracies across all datasets (e.g., 33.56 – 78.99% on Glass) and higher average errors, indicating a limited capacity for handling complex or imbalanced data. BatClassifier shows moderate performance with variable execution times, while GAClassifier achieves consistent accuracies but generally requires significantly longer computation times than SAClassifier or SGD. These findings emphasize the practical trade-off: PSO and Tabu are preferable for maximum accuracy on solvable tasks, while SAClassifier offers the best balance of speed, generalization, and accuracy relative to the gradient-based baseline.

Visual inspection of convergence behaviors (Figures 2, 3, and 4) further corroborates these results. PSOClassifier and SAClassifier demonstrate rapid convergence on simpler datasets such as Iris, while ACOClassifier exhibits slower convergence consistent with its elevated error rates. These patterns suggest that classifier performance is influenced by both the choice of metaheuristic and dataset characteristics, including dimensionality and class balance.

To support these findings, t-test was employed to statistically compare the performances of classifiers on datasets, as indicated in Table 10. Significant differences in performance are confirmed by low p-values (e.g., $4.4e^{-39}$ of SAClassifier vs. ACOClassifier on Breast Cancer) and similar performance was demonstrated by higher p-values (e.g. $2.5e^{-2}$ of TabuClassifier vs. PSOClassifier on Breast Cancer). To further confirm these results across all datasets, we used the non-parametric Friedman test that produced a test value of 23.21 with a p-value of 0.0007 ($p < 0.001$) hence rejecting the null hypothesis and realizing that the difference in the performance of the algorithms was statistically significant. A Nemenyi post-hoc test was then used to determine particular differences between pairs. The findings have indicated that even though the metaheuristics are statistically similar in terms of rankings (e.g., $p = 0.944$ in case of SAClassifier vs. TabuClassifier), strong differences can be observed between the best-performing metaheuristics (GA, PSO) and the Backpropagation baseline ($p = 0.001$). This statistical validation at multiple levels enhances the integrity of the comparative analysis, which becomes the basis of deciding on the right metaheuristics depending on the specifics of the datasets and realistic optimization objectives.

5.3. Practical Recommendations and Trade-offs

It is important to note as demonstrated in the experimental results that there is no single metaheuristic which can be applied as a universal solution in training neural networks. We give a qualitative comparison of trade-offs to help practitioners choose the right algorithm in Table 13.

- Precision vs. Overfitting: Although PSO and Tabu Search always recorded the highest training accuracies, they tended to overfit significantly with large generalization gaps ($> 20\%$ on Heart Disease). These algorithms are most applicable to problems whose training data is exhaustive of the problem space or, to problems where strict regularization mechanisms are employed. Simulated Annealing (SA) in contrast has a better balance, frequently attaining competitive testing performance with only slight overfitting, and is more reliable to use in tasks where the data is limited.

- Computational Cost vs. Scalability : In resource-constrained settings, SA is the clear choice, as it can achieve comparable execution times with good testing performance with only minor overfitting, and it is less sensitive to local optima traps, as was used in SGD. Population-based techniques (GA, PSO, Bat) are much more costly

in terms of computation since they can consider many agents at a time. This is a scaling limitation that is severe when the dimensionality of the problems is high.

Table 13. Comparative summary of algorithmic trade-offs for Neural Network Training.

Algorithm	Training Accuracy	Generalization	Convergence Speed	Scalability	Recommended Use Case
SA (Simulated Annealing)	Moderate	**High** (Low Overfitting)	**Very Fast**	High	Real-time apps, Low-resource hardware
PSO (Particle Swarm)	**Very High**	Low (High Overfitting)	Moderate	Low	Complex, non-convex landscapes (requires regularization)
Tabu Search	**Very High**	Low (High Overfitting)	Moderate	Low	Problems requiring intense local exploitation
GA (Genetic Algorithm)	High	Moderate	Slow	Very Low	Global search where time is not critical
Bat Algorithm	Moderate	Moderate	Variable	Low	Alternative to PSO for multimodal problems
ACO (Ant Colony)	Low	Low	Moderate	Very Low	Not recommended for standard MLP training
SGD / Backprop (Baseline)	Low/Moderate	High	Fast	**Very High**	Convex problems, Large-scale Deep Learning

6. Conclusion

This study systematically evaluated the performance of several metaheuristic algorithms for training feedforward neural networks across a diverse set of benchmark datasets. The comparative analysis highlighted that Particle Swarm Optimization and Simulated Annealing consistently offered competitive results in terms of classification accuracy, convergence behavior, and computational efficiency, while certain methods, such as Ant Colony Optimization, exhibited limitations in generalization performance across multiple datasets. These findings reinforce the potential of metaheuristics as viable optimization strategies for neural network training, particularly in scenarios where gradient-based methods face challenges such as local minima or non-differentiable cost functions.

Despite the encouraging results, this work also reveals opportunities for further research. First, the established benchmark framework can be extended to evaluate other neural network architectures, such as Convolutional Neural Networks (CNNs), to assess the scalability and adaptability of these metaheuristics to deep learning contexts. Second, future experiments should explore more complex datasets with higher dimensionality and non-linear dependencies to better reflect real-world application challenges. Finally, investigating alternative network topologies and structural adaptations could yield architectures more naturally suited to specific problem domains, thereby enhancing both predictive performance and computational efficiency.

Through these extensions, the current benchmarking approach can evolve into a more comprehensive evaluation platform, contributing to the design of robust, adaptable, and high-performing neural network solutions optimized via metaheuristics.

Acknowledgement

This work was supported by the National Center for Scientific and Technical Research (CNRST), Rabat, Morocco, as part of the "PhD-ASsociate Scholarship – PASS" program, and using computational resources of HPC-MARWAN (hpc.marwan.ma) provided by the National Center for Scientific and Technical Research (CNRST), Rabat, Morocco.

REFERENCES

1. W.S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics, 5 (4) (1943) 115–133.
2. F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, 65 (6) (1958) 386–408.
3. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (6088) (1986) 533–536.

4. G. E. Hinton, S. Osindero and Y. W. Teh *A fast learning algorithm for deep belief nets. Neural computation* , JAMA, 18 (7) (2006) 1527-1554 .
5. A. Espinal, M. Sotelo-Figueroa, J.A. Soria Alcaraz, M. Ornelas, H. Puga, M. Carpio, R. Baltazar, and J.L. Rico, *Comparison of PSO and DE for Training Neural Networks*, 10th Mexican International Conference on Artificial Intelligence, 2 (2011) 67–72.
6. T. Si, S. Hazra, and N.D. Jana, *Artificial Neural Network Training Using Differential Evolutionary Algorithm for Classification*, Proceedings of the International Conference on Information Systems Design and Intelligent Applications, 132 (2012) 769–778.
7. M. Carvalho and T.B. Ludermir, *An Analysis of PSO Hybrid Algorithms For Feed-Forward Neural Networks Training*, Proceedings of the Ninth Brazilian Symposium on Neural Networks (SBRN 2006), (2006).
8. A. Al Kawam and N. Mansour, *Metaheuristic Optimization Algorithms for Training Artificial Neural Networks*, International Journal of Computer and Information Technology, 1 (2) (2012).
9. M. Sewak, S.K. Sahay, and H. Rathore, *An Overview of Deep Learning Architecture of Deep Neural Networks and Autoencoders*, Journal of Computational and Theoretical Nanoscience, 17 (1) (2020) 182–188.
10. X. Glorot, A. Bordes, and Y. Bengio, *Deep Sparse Rectifier Neural Networks*, Proceedings of Machine Learning Research, 15 (2011) 315–323.
11. K. Janocha and W.M. Czarnecki, *On Loss Functions for Deep Neural Networks in Classification*, Schedae Informaticae, 25 (2016) 49–59.
12. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, *Optimization by Simulated Annealing*, Science, 220 (4598) (1983) 671–680.
13. M. Dorigo and G. Di Caro, *Ant colony optimization: a new meta-heuristic*, Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), IEEE, (1999) 1470–1477.
14. X.-S. Yang, *A New Metaheuristic Bat-Inspired Algorithm*, Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), 284 (2010) 65–74.
15. N. Adil and H. Lakhbab, *A new modified bat algorithm for global optimization*, RAIRO-Operations Research, 57 (5) (2023) 2659–2685.
16. F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, 13 (5) (1986) 533–549.
17. J. Kennedy and R. Eberhart, *Particle swarm optimization*, International Conference on Neural Networks, IEEE, (1995) 1942–1948.
18. J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology*, University of Michigan Press, Ann Arbor, MI, (1975).
19. H. Hofmann, *Statlog (German Credit Data)*, UCI Machine Learning Repository, (1994).
20. R.A. Fisher, *Iris*, The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7 (2) (1936) 179-188.
21. W.H. Wolberg, *Breast Cancer Wisconsin (Prognostic)*, UCI Machine Learning Repository, (1995).
22. P. Cortez, *Wine Quality*, UCI Machine Learning Repository, (2009).
23. A. Janosi, *Heart Disease*, UCI Machine Learning Repository, (1989).
24. B. German, *Glass Identification*, UCI Machine Learning Repository, (1987).
25. F.Z. El-Hassani, M. Amri, N.E. Joudar, and K. Haddouch, *A New Optimization Model for MLP Hyperparameter Tuning: Modeling and Resolution by Real Coded Genetic Algorithm*, Neural Processing Letters, 56 (2) (2024) 105.
26. C. Guotai, M.Z. Abedin, and F.E. Moula, *Modeling Credit Approval Data With Neural Networks: An Experimental Investigation and Optimization*, Journal of Business Economics and Management, 18 (2) (2017) 224–240.
27. M.H. Alshayeji, H. Ellethy, S. Abed, and R. Gupta, *Computer-aided detection of breast cancer on the Wisconsin dataset: An artificial neural networks approach*, Biomedical Signal Processing and Control, 71 (2022) 103141.
28. M.J. El-Khatib, B.S. Abu-Nasser, and S.A. Naser, *Glass Classification Using Artificial Neural Network*, International Journal of Academic Pedagogical Research, 3 (2) (2019) 25–31.
29. H. Bihri, R. Nejjari, S. Azzouzi, and M.E.H. Charaf, *An Artificial Neural Network-Based System to Predict Cardiovascular Disease*, Advances in Information, Communication and Cybersecurity, 357 (2022).