STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **12**, September 2024, pp 1409–1424.
Published online in International Academic Press (www.IAPress.org)

A novel Mathematical Modeling for Deep Multilayer Perceptron Optimization: Architecture Optimization and Activation Functions Selection

Taoufyq Elansari*, Mohammed Ouanan, Hamid Bourray

TSI Team, Department of Computer Sciences, Faculty of Sciences, Moulay Ismail University, B.P. 11201 Zitoune, Meknes, Morocco

Abstract The Multilayer Perceptron (MLP) is an artificial neural network composed of one or more hidden layers. It has found wide use in various fields and applications. The number of neurons in the hidden layers, the number of hidden layers, and the activation functions employed in each layer significantly influence the convergence of MLP learning algorithms. This article presents a model for selecting activation functions and optimizing the structure of the multilayer perceptron, formulated in terms of mixed-variable optimization. To solve the obtained model, a hybrid algorithm is used, combining stochastic optimization and the backpropagation algorithm. Our algorithm shows better complexity and execution time compared to some other methods in the literature, as the numerical results show.

Keywords Multilayer Perceptron, Supervised Training, Mixed-variable optimization problems, Stochastic optimization.

AMS 2010 subject classifications 34A34, 65L05.

DOI: 10.19139/soic-2310-5070-1990

1. Introduction

Multilayer Perceptron Neural Networks (MLPNNs) stand as enduring pillars in the realm of artificial intelligence and deep learning, serving as classifiers with a myriad of advantages. Their inherent versatility makes MLPs applicable across a broad spectrum of tasks, ranging from image classification to natural language processing. What sets MLPs apart is their straightforward architecture, comprising input layers, hidden layers, and an output layer. They also serve as models for comparison, providing a benchmark against which the performance of more complex architectures, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) [1, 2], can be evaluated. MLPs seamlessly integrate with other cutting-edge architectures, like linking in the final process of CNNs for classification [3, 4] or being embedded within the intricate frameworks of GANs to map random noise to data distribution. MLPs demonstrate their adaptability and collaboration with other models.

The development of MLPNNs encounters two significant challenges: learning and architecture selection. Learning involves minimizing the error function between the desired and MLPNN output, employing algorithms like backpropagation [5], metaheuristics [6, 5], and Bayesian learning methods [7, 8]. The second challenge is selecting the optimal model, defined by hyper-parameters. Constructing an efficient MLPNN model involves algorithm selection and hyper-parameter tuning [9]. Model complexity depends on hidden layers and neurons, crucial hyper-parameters [10], adjusted based on dataset complexity. Balancing between underfitting and overfitting is essential, requiring optimal hidden layers and neurons [11]. To tackle the issue of hyperparameter optimization, various methods have been suggested, such as grid search [17], random search [18], Bayesian optimization [19], and genetic algorithms [13]. Although these methods have shown effectiveness in specific scenarios, they often

*Correspondence to: Taoufyq Elansari (Email: t.elansari@edu.umi.ac.ma). TSI Team, Department of Computer Sciences, Faculty of Sciences, Moulay Ismail University, B.P. 11201 Zitoune, Meknes, Morocco.

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

come with high computational costs and may not be ideal for optimizing real-valued hyperparameters and large-scale neural networks [14].

This paper proposes a new mathematical optimization model for selecting activation functions and optimizing the structure of the multilayer perceptron. The optimization model is designed to receive the maximum architecture and then remove unnecessary hidden layers and neurons. This approach is superior to traditional experimental methods for selecting architecture, which are often time-consuming and can lead to overfitting or underfitting. The proposed model is a nonlinear optimization problem with mixed variables. To solve this model, a hybrid algorithm based on stochastic optimization and backpropagation is applied. The use of stochastic optimization helps to avoid getting trapped in local optima and allows for a balance between model accuracy and complexity. The hybrid algorithm combines the strengths of both optimization methods, resulting in faster convergence and better performance. The impetus behind this endeavor lies in expediting MLP architecture selection processes, with the overarching goal of circumventing time-consuming procedures prone to overfitting and underfitting, thus elevating efficiency and efficacy in deep learning applications.

The paper adopts the following structure: Related Work, in Section 2. The proposed optimization model for multilayer perceptron in section 3. Solving the optimization model obtained in section 5. The experimental study and discussion of the results are presented in the last section.

2. Related Work

2.1. Machine learning hyperparameter optimisation

In machine learning models, there are two types of parameters: Model parameters, which can be initialized and updated through data learning operations (e.g., weights of MLP), and hyperparameters, which define the model architecture and must be determined before the model training, as they cannot be estimated directly from the data. Typically, developing an effective machine learning model is a complex and time-consuming endeavor that involves identifying the appropriate algorithm and fine-tuning its hyperparameters to achieve an optimal model structure. The classical method is not effective for many problems due to factors such as a large number of hyperparameters, complex models, time-consuming evaluations, and nonlinear interactions between hyperparameters.

The main goal of HPO is to automate the hyperparameter configuration process and facilitate the successful application of machine learning models to practical problems [15]. One of the benefits of using automated hyperparameter tuning is that it reduces the manual effort required, which is particularly useful for ML researchers working with large datasets or complex algorithms that have many hyperparameters to adjust [16]. This can free up researchers' time to focus on other aspects of the project. Additionally, automated hyperparameter tuning can lead to improved performance of ML models. To obtain the optimal hyperparameters that give the best performance on different datasets or problems, ultimately resulting in more accurate and efficient ML models, some methods are proposed in the literature. Grid search [17] and random search [18] are decision theory-based methods used to optimize hyperparameters. Grid search exhaustively searches for the optimal configuration in a predefined space of hyperparameters, while random search randomly selects combinations of hyperparameters within limited runtime and resources. Bayesian optimization models [19], unlike grid search and random search, use previous results to determine the next hyperparameter setting, thereby reducing the need for unnecessary estimations and resulting in faster convergence towards the optimal hyperparameter combination. Successive Halving [20] is an optimization algorithm that randomly samples hyperparameter configurations, evaluates their performance, and eliminates the worst-performing configurations until only one remains. Hyperband [21] is an extension of the successive halving algorithms introduced by Lisha Li and others.

2.2. MLPNN hyperparameter optimisation using metaheuristics

Metaheuristic algorithms are a set of techniques used to solve complex and non-convex optimization problems with a large search space, including HPO problems. Two popular metaheuristic algorithms used for HPO problems are the genetic algorithm (GA) and particle swarm optimization (PSO).

Several studies have utilized Genetic Algorithms (GAs) to optimize the structure and hyperparameters of multilayer perceptron neural networks. In this study [22], a GA was applied to optimize the structure of MLPNNs, including the number of hidden layers, neurons per layer. The GA evolved a population of MLPNN structures through selection, crossover, and mutation. Fitness evaluation was based on MLPNN performance. Another work [23] focused on GA-based optimization of hyperparameters like learning rate, momentum, and weight decay. The GA searched for optimal hyperparameter combinations to maximize MLPNN performance. Similarly, a study optimized MLPNN hyperparameters for financial distress prediction using a GA [24], considering hidden layers, neurons, learning rate, and momentum. The GA improved performance by evolving hyperparameter combinations. Particle Swarm Optimization (PSO) has been successfully used to optimize hyperparameters in MLP neural networks. Studies such as [25] and [26] applied PSO to find optimal combinations of hyperparameters like learning rate, momentum, and hidden neurons or layers. Each particle in the swarm represented a unique hyperparameter combination, and its position was updated based on its own and global best solutions. Another study [27] optimized both structure and hyperparameters using PSO, simultaneously exploring the number of layers, neurons, and hyperparameters. Another work [28] use an algorithm combining Tabu search and Gradient descent to automatically optimize the number of hidden layers and neurons, improving generalization ability. Nonetheless, these approaches lack effectiveness in the optimization of activation function selection. In this paper we aim to introduce a novel optimization model that focuses on enhancing the selection process of both architectural design and activation functions through the use of combinatorial optimization techniques.

2.3. Multilayer perceptron: Architecture and Training

2.3.1. Multilayer perceptron architecture The multi-layer perceptron neural network consists of multiple layers, including an input layer, one or more hidden layers, and an output layer [29]. The MLPNN is a feedforward neural network that processes information in one direction only, from the input layer to the output layer. The input layer corresponds to the network's inputs and has the same number of neurons as the number of input variables. The output layer corresponds to the network's outputs and has the same number of neurons as the number of output classes. The architecture of MLP is depicted in Figure 1.

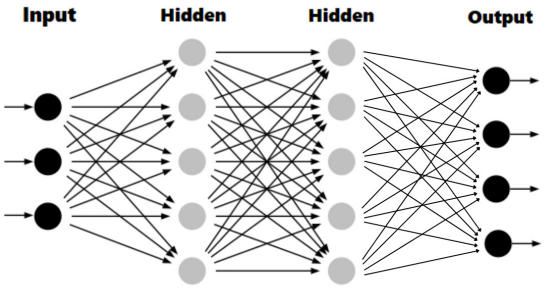

Figure 1. Example of multilayer feed forward neural network

2.3.2. Learning techniques And Backpropagation The main goal of learning is to create a technique that allows the network to adapt its parameters based on the training examples it receives. There are three primary types of learning: supervised, unsupervised, and hybrid. Learning refers to the process of tweaking a system's parameters to generate the desired response to a given input or stimulus [30]. One type of learning, supervised learning, is employed by the multilayered perceptron.

The backpropagation learning algorithm [31] is an iterative optimization method that seeks to determine the connection weights that minimize the mean square error incurred by the network on the training set. It is based on the gradient descent approach [32]. During backpropagation learning, a vector is fed from the training set into the network input. The output of the network is then computed by propagating through the layers from the input layer to the output layer via hidden layers. Comparing the resulting output to the desired output produces an error. The gradient of this error can be calculated and propagated from the output layer to the input layer, hence the term

"backpropagation." This enables the modification of the network's weighted connections and facilitates learning. This process is repeated for each vector from the training set until the stopping criteria are met.

3. The Proposed Optimization Model for Multilayer Perceptron

In this section, we propose a mathematical modeling of the multilayer perceptron for architecture optimization and activation function selection. The notions used in this mathematical modeling are as follows:

- J **:** Size of training data set.
- X **:** Training data set as :
  $X = \left\{(x^h, Y^h); h \in \{1, \ldots, J\}\right\}$ where Y^h desired output and x^h input data.
- N **:** The number of hidden layers in the MLPNN.
- $N+1$ **:** The index of the output layer.
- n_0 **:** The number of neurons in input layer.
- n **:** Initial number of neurons in each hidden layer.
- m **:** The number of neurons in output layer.
- K **:** Number of activation functions suggested.
- $\hat{Y}$ **:** The output of MLPNN.
- a^k : The output of the hidden layer for $k = 1, ..., N$.
- $w_{i,j}^k$ **:** The weighted connections that connect neuron i of layer k to neuron j of layer $k-1$.
- f **:** Activation function.
- u_i^k **:** Binary variable as: $u_i^k = \begin{cases} 1 & \text{if the } i^{th} \text{ neuron in the } k^{th} \text{ hidden layer is used.} \\ 0 & \text{if the } i^{th} \text{ neuron is deleted} \end{cases}$
- v_i **:** Binary variable as: $v_i = \begin{cases} 1 & \text{if the } i^{th} \text{ hidden layer is used.} \\ 0 & \text{if the } i^{th} \text{ hidden layer is deleted} \end{cases}$
- α_i^k **:** Binary variable as: $\alpha_i^k = \begin{cases} 1 & \text{if the } i^{th} \text{ activation function is used in the } k^{th} \text{ layer.} \\ 0 & \text{otherwise} \end{cases}$

Activation Function Identification In order to improve the performance of a neural network, it is indeed important to consider different activation functions for each layer. To incorporate multiple activation functions within the same network, you can assign a different activation function to each layer. By doing so, you allow the network to learn and represent complex patterns and relationships more effectively. For the use of mixed activation functions in MLPNN, we use the following expressions:

$$f^k(x) = \sum_{i=1}^{K} \alpha_i^k f_i(x) \quad \forall k \in \{1, \ldots, N+1\} \tag{1}$$

where: $\sum_{i=1}^{K} \alpha_i^k = 1$ and $\alpha_i^k \in \{0,1\}\ \forall i \in \{1, \ldots, K\} \quad \forall k \in \{1, \ldots, N+1\}$

Output of first hidden layer: The first hidden layer neurons are connected directly to the input layer of the MLPNN. Consider an input $x^h = (x_1, x_2, ..., x_{n_0})$, the output of the first hidden layer is calculated as shown below:

$$\mathrm{a}^1 = \begin{pmatrix} \mathrm{a}_1^1 \\ \vdots \\ \mathrm{a}_j^1 \\ \vdots \\ \mathrm{a}_n^1 \end{pmatrix} = \begin{pmatrix} u_1^1 f^1\left(\sum_{i=1}^{n_0} w_{1,i}^1 x_i\right) \\ \vdots \\ u_j^1 f^1\left(\sum_{i=1}^{n_0} w_{j,i}^1 x_i\right) \\ \vdots \\ u_n^1 f^1\left(\sum_{i=1}^{n_0} w_{n,i}^1 x_i\right) \end{pmatrix} \tag{2}$$

Output of the hidden layers: To calculate the output of each neuron of the k^{th} hidden layer, where $k = 1, \ldots, N$, we propose this expression:

$$\begin{pmatrix} \mathrm{a}_1^k \\ \vdots \\ \mathrm{a}_j^k \\ \vdots \\ \mathrm{a}_n^k \end{pmatrix} = (1 - v_k)\, \mathrm{a}^{k-1} + v_k \begin{pmatrix} u_1^k f^k \left(\sum_{i=1}^n w_{i1}^k \mathrm{a}_i^{k-1}\right) \\ \vdots \\ u_j^k f^k \left(\sum_{i=1}^n w_{ij}^k \mathrm{a}_i^{k-1}\right) \\ \vdots \\ u_n^k f^k \left(\sum_{i=1}^n w_{in}^k \mathrm{a}_i^{k-1}\right) \end{pmatrix} \tag{3}$$

Output of the output Layer: The output of the MLPNN is determined by the expressions below:

$$\hat{Y}(U, V, x^h, W, \alpha) = \begin{pmatrix} \hat{Y}_1 \\ \vdots \\ \hat{Y}_i \\ \vdots \\ \hat{Y}_m \end{pmatrix} = \begin{pmatrix} f^{N+1}\left(\sum_{j=1}^n w_{j1}^{N+1} \mathrm{a}_j^N\right) \\ \vdots \\ f^{N+1}\left(\sum_{j=1}^n w_{ji}^{N+1} \mathrm{a}_j^N\right) \\ \vdots \\ f^{N+1}\left(\sum_{j=1}^n w_{jm}^{N+1} \mathrm{a}_j^N\right) \end{pmatrix} \tag{4}$$

3.1. Objective function

The objective function is the average error calculated between the obtained output and the desired output for all patterns in the training set:

$$G(U, V, X, W, \alpha) = \frac{1}{J} \sum_{h=1}^{J} \| \hat{Y}(U, V, x^h, W, \alpha) - Y^h \|_2^2 \tag{5}$$

3.2. Constraints

Constraints guarantee the existence of the hidden layers.

$$\sum_{j=1}^{N} v_j \geqslant 1 \tag{6}$$

These constraints guarantees the existence of neurons in the active hidden layers.

$$\sum_{i=1}^{N} v_i \times \prod_{j=1}^{n} \left(1 - u_j^i\right) = 0 \tag{7}$$

Constraints guaranteed if a hidden layer is not active then all the neurons of this layer are not active.

$$\sum_{i=1}^{N} (1 - v_i) \sum_{j=1}^{n} u_j^i = 0 \tag{8}$$

The constraint below ensure that neural network is assigned one activation functions:

$$\sum_{i=1}^{K} \alpha_i^k = 1 \quad \forall\, k \in \{1, \ldots, N+1\} \tag{9}$$

3.3. Optimization Model

The proposed combinatorial optimization model for optimizing multilayer perceptron architecture and selecting activation functions:

$$
(P)\begin{cases}
\min \frac{1}{2J}\sum_{i=1}^{J}\left\|\hat{Y}(V,U,x^i,\alpha,W)-Y^i\right\|_2^2 \\
\text{Subject to} \\
\sum_{j=1}^{N} v_j \geqslant 1 \\
\sum_{i=1}^{N} v_i \times \prod_{j=1}^{n}\left(1-u_j^i\right)=0 \\
\sum_{i=1}^{N}(1-v_i)\sum_{j=1}^{n} u_j^i=0 \\
\sum_{i=1}^{K}\alpha_i^k=1 \quad \forall\, k\in\{1,\ldots,N+1\} \\
W=\left(w_{i,j}^k\right)_{\substack{1\leq j\leq n_k\\ 1\leq i\leq n_{k-1}\\ 1\leq k\leq N+1}} \qquad \text{where } w_{i,j}^k\in\mathbb{R} \\
U=\left(u_i^k\right)_{\substack{1\leq i\leq n\\ 1\leq k\leq N}} \qquad \text{where } u_i^k\in\{0,1\} \\
V=(v_k)_{1\leq k\leq N} \qquad \text{where } v_k\in\{0,1\} \\
\alpha=\left(\alpha_t^k\right)_{\substack{1\leq t\leq K\\ 1\leq k\leq N+1}} \qquad \text{where } \alpha_t^k\in\{0,1\}
\end{cases}
$$

Figure 2 illustrates an example of the multi-layer perceptron neural network architecture both before and after undergoing training.

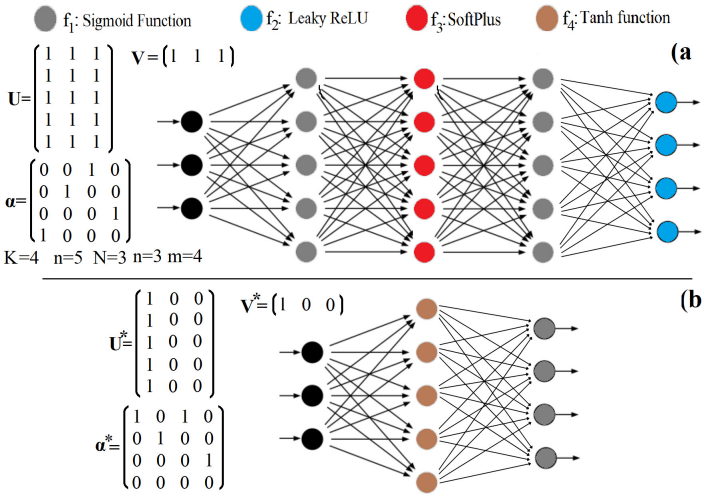

Figure 2. MLPNN architecture before and after training

4. Solving the optimization model obtained

In this algorithm, we search for an optimal architecture by stochastic optimization and adjust the weighted connections of the multi-layer perceptron neural network by the backpropagation algorithm.

4.1. Notations

We calculate the error of the layers of the network by the following formulas:

- To calculate the error for the output layer:

$$e^{N+1} = \begin{pmatrix} e_1^{N+1} \\ \vdots \\ e_k^{N+1} \\ \vdots \\ e_m^{N+1} \end{pmatrix} = \begin{pmatrix} f'^{N+1}\left(h_1^{N+1}\right)\left(\hat{Y}_1 - Y_1\right) \\ \vdots \\ f'^{N+1}\left(h_k^{N+1}\right)\left(\hat{Y}_k - Y_k\right) \\ \vdots \\ f'^{N+1}\left(h_m^{N+1}\right)\left(\hat{Y}_m - Y_m\right) \end{pmatrix} \tag{10}$$

- To calculate the error for the N^{th} hidden layer:

$$e^{N} = \begin{pmatrix} e_1^{N} \\ \vdots \\ e_k^{N} \\ \vdots \\ e_n^{N} \end{pmatrix} = \begin{pmatrix} f'^{N}\left(h_1^{N}\right)\sum_i w_{i1}^{N+1} e_i^{N+1} \\ \vdots \\ f'^{N}\left(h_k^{N}\right)\sum_i w_{ik}^{N+1} e_i^{N+1} \\ \vdots \\ f'^{N}\left(h_n^{N}\right)\sum_i w_{in}^{N+1} e_i^{N+1} \end{pmatrix} \tag{11}$$

- To calculate the error of the l^{th} hidden layer for $l \in \{N-1, ..., 1\}$, we use the formula below:

$$e^{l} = \begin{pmatrix} e_1^{l} \\ \vdots \\ e_k^{l} \\ \vdots \\ e_n^{l} \end{pmatrix} = (1 - v_l)\, e^{l+1} + v_l \begin{pmatrix} u_1^l f'^{l}\left(h_1^{l}\right)\sum_i^n w_{i1}^{l+1} e_i^{l+1} \\ \vdots \\ u_k^l f'^{l}\left(h_k^{l}\right)\sum_i^n w_{ik}^{l+1} e_i^{l+1} \\ \vdots \\ u_n^l f'^{l}\left(h_n^{l}\right)\sum_i^n w_{in}^{l+1} e_i^{l+1} \end{pmatrix} \tag{12}$$

The symbols that we will use in the stochastic optimization method are defined by the following notions:

- $MaxItr$: the number of iterations.
- T : system temperature.
- $\Delta F = F(U', X, C', W') - F(U, X, C, W)$: system energy.
- $X = (x_h, y_h)_{1 \leq h \leq J}$: training data.
- α: scale temperature reduction ($\alpha < 1$).
- μ: represents the learning rate (low magnitude, between μ_{min} and μ_{max})
- $rand(0, 1)$: operator returns a real value between 0 and 1 randomly according to a uniform law.
- $randInt(1, n)$: operator returns a number between 1 and n randomly according to a uniform law.

4.2. The proposed method

The steps of the algorithm are as follows:

1. From the hyperparameter search space, the MLP with the largest structure will be selected as the initial model.

2. The structure of this selected model will be modified to create a new MLP. This modification can involve changes such as adjusting the number of layers, the number of neurons in each layer, or the activation functions.
3. Gradient backpropagation iterations will be performed for both the initial model and the modified model. This involves propagating the input data forward through the network, calculating the loss, and then backpropagating the gradients to update the weights and biases of the MLP using an optimization algorithm.
4. After each iteration, the performance of both models will be evaluated based on metrics such as accuracy or error. The model with the best performance or the lowest error will be retained.
5. If the maximum number of iterations specified for the algorithm is reached, the algorithm terminates. Otherwise, the process returns to step 2, where the structure of the retained model is modified again to create a new MLP. This iterative process continues until the maximum number of iterations is reached or another stopping criterion is met.

The proposed algorithm is presented as follows (Algorithm 1):

Algorithm 1 The proposed method.

Require: $(U(0), V(0), W(0), \alpha(0))$, $X = (x_h, y_h)_{0 \leq h \leq J}$,$\mu_{min}$,$\mu_{max}$ and IterMax.

$t = 0$
$(U^*, V^*, W^*, \alpha^*) = (U(0), V(0), W(0), \alpha(0))$
while t < IterMax **do**
  $W = W(t)$
  $k = rand(1, N)$
  $v_k = (1 - v_k(t))$
  $s = 0$
  while $s < max(1, T)$ **do**
    $q = rand(1, n)$
    $u_k^q = (1 - u_k^q(t))$ and $s = s + 1$
  end while
  $v_k = 1 - \prod_{j=1}^{n} \left(1 - u_j^k\right)$
  $s = rand(1, K)$ and $r = rand(1, N + 1)$
  $\alpha^r(t) = (0, \ldots, 0)$ and $\alpha_s^r(t) = 1$
  $\mu = rand(\mu_{min}, \mu_{max})$
  for $(x, y) \in X$ **do**
    Calculate: $F(U, V, x, W, \alpha)$ and $F(U(t), V(t), x, W(t), \alpha(t))$.
    Calculate: e^l and $e^l(t)$,$\forall l \in \{N + 1, \ldots, 1\}$.
    We update the weights in all layers:$w_{ij}^l = w_{ij}^{(l)} - \mu e_i^{(l)} \mathrm{a}_j^{(l-1)}$.
    We update the weights in all layers: $w_{ij}^l(t) = w_{ij}^{(l)}(t) - \mu e_i^{(l)}(t) \mathrm{a}_j^{(l-1)}(t)$.
  end for
  Calculate $G(U, V, X, W, \alpha)$ and $G(U(t), V(t), X, W(t), \alpha(t))$
  Calculate $\Delta G = G(U, V, X, W, \alpha) - G(U(t), V(t), X, W(t), \alpha(t))$
  if $\Delta G \leq 0$ **then**
    $(U(t + 1), V(t + 1), W(t + 1), \alpha(t + 1)) = (U, V, W, \alpha)$
  end if
  $t = t + 1$ and $T = \alpha T$
  if $G(U(t), V(t), X, W(t), \alpha(t)) < G(U^*, V^*, X, W^*, \alpha^*)$ **then**
    $(U^*, V^*, W^*, \alpha^*) = (U(t), V(t), W(t), \alpha(t))$
  end if
end while

Ensure: $(U^*, V^*, W^*, \alpha^*)$.

Our hybrid algorithm proposes a novel approach to optimizing multilayer perceptron (MLP) architectures. By combining gradient descent with stochastic optimization, it efficiently explores the hyperparameter space. It starts with the largest MLP structure and iteratively modifies it by adjusting layers, neurons, and activation functions. Random selection and mutation rates ensure diverse exploration, aiding in escaping local optima. Gradient backpropagation updates model parameters, optimizing them with respect to the objective function. The process continues until a termination criterion is met. Through extensive testing, our algorithm has shown promise in enhancing MLP performance across various deep learning tasks.

5. Implementation and numerical results

The purpose of this section is to illustrate the performance and advantages of our model, for this we developed the programs in python and tested them in the AMD Ryzen 5 5600H with Radeon Graphics processor at 3.30 GHz and 16 GB of RAM.

5.1. Classification problems

5.1.1. Datasets To illustrate the advantages of our model and the proposed method for obtaining an approximate solution to this model, we conducted tests on classification problems using siven datasets from the UCI machine learning repository [33], selected for computational experiments. This selection included datasets of varying nature, ranging from small datasets like Iris to larger datasets such as Thyroid and Spambase. The class distribution also varied, including equilibrium datasets like Iris or Seeds.

In Table 1, the details of the datasets are presented, including the number of examples, the number of attributes, and the class.

Table 1. Characteristic of the data set used.

Database	Examples	Attributes	Real	Integer	Nominal	Class
Iris	150	4	4	0	0	3
Seeds	210	7	7	0	0	3
Wine	178	13	13	0	0	3
Thyroid	7200	21	6	15	0	3
Heart statlog	270	13	1	12	0	2
WDBC	768	30	30	0	0	2
Spambase	4601	57	55	2	0	2

5.1.2. Performance measurement In this study, all datasets were divided for training and testing by 50% and 50%, respectively. We utilized precision measure extracted from the confusion matrix to evaluate the classification performance in the test set.

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{13}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{14}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{15}$$

$$F1\_Score = 2 \times \left(\frac{(Precision \times Recall)}{(Precision + Recall)} \right) \tag{16}$$

Where TN indicates true negatives, TP indicates true positives, FN indicates false negatives, and FP indicates false positives.

We calculate the number of neurons in the hidden layers by:

$$n_{\mathrm{N}} = \sum_{k=1}^{N} \sum_{j=1}^{n} u_j^k \tag{17}$$

5.2. Classification Results

Figure 3 illustrates the variation in the number of neurons in the hidden layers as iterations progress, considering different initial architectures for all the databases. The graph visually represents the current optimal solution for the number of neurons in the hidden layers at each iteration.

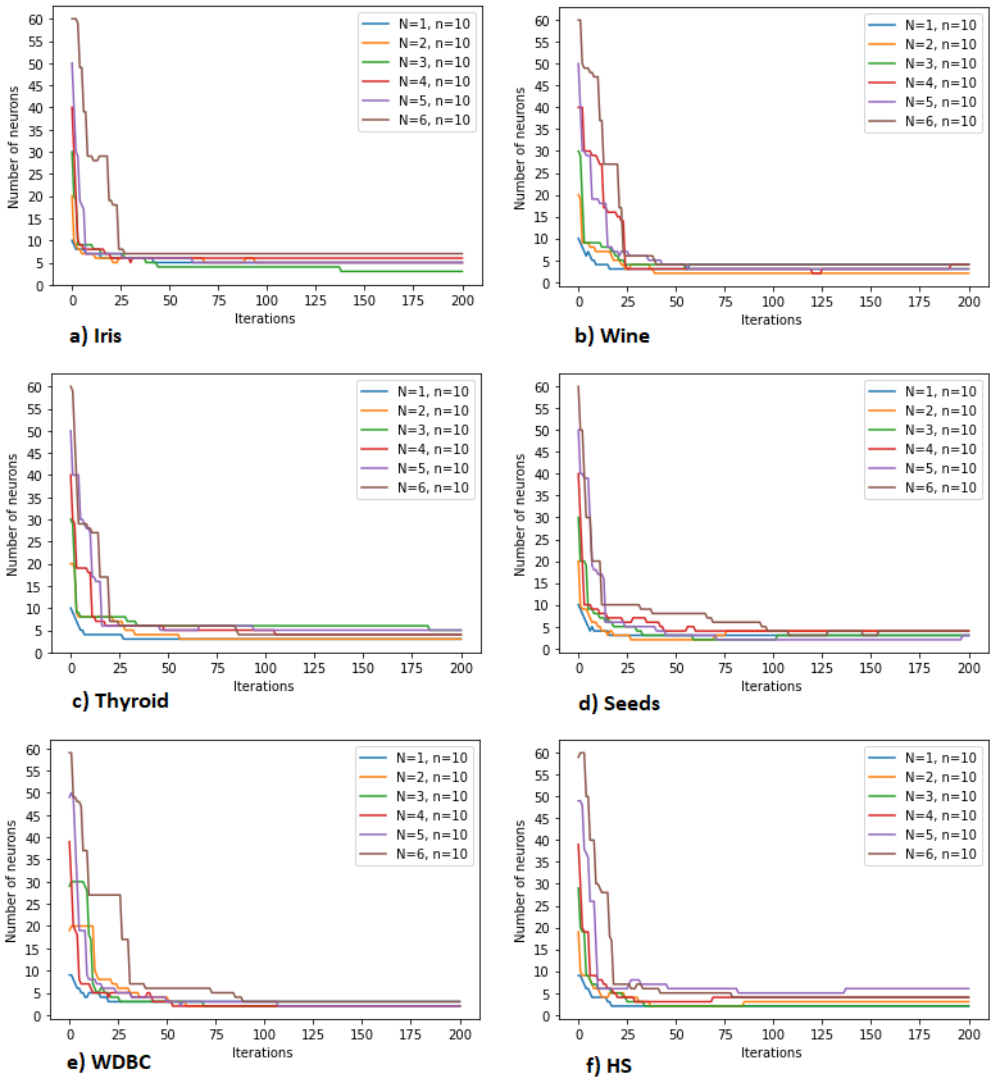

Figure 3. Optimal architecture for the all datasets with different initial numbers of neurons.

The analysis of Figure 3 reveals an interesting trend: regardless of the initial architecture used for the neural network, all the curves converge towards a narrow range or interval of the number of neurons in the hidden layers. This convergence indicates the stability of the algorithm employed, even when applied to various databases. The fact that different initial architectures lead to similar outcomes suggests that the algorithm is robust and capable of finding an optimal configuration for the hidden layers. This stability is an encouraging sign, as it implies that the algorithm is not overly sensitive to the specific initial conditions or network configurations. Such stability across different databases is valuable because it indicates that the algorithm can effectively adapt to diverse datasets and generalize well to new, unseen data. This characteristic is essential for ensuring the reliability and versatility of the neural network model.

Table 2 shows the pre-training architecture and the optimal post-training architecture, the classification error of the test set and the classification error of the training set for each database, obtained by solving our optimization problem 3.3 with Algorithm 1.

Table 2. Initials and optimized architectures.

Dataset	Maximum architectures	Optimized architectures	L.Accuracy	T.Accuracy
Iris	(4,20,20,20,3)	(4,4,3)	98.66	97.33
Wine	(13,20,20,20,3)	(13,4,3)	100.0	98.88
Seeds	(7,20,20,20,3)	(7,4,3)	96.00	95.95
Thyroid	(21,20,20,20,3)	(21,4,3)	95.33	95.13
WDBC	(30,20,20,20,2)	(30,7,2)	98.48	93.44
Heart statlog	(13,20,20,20,2)	(13,7,2)	92.00	90.00

In Table 2, it is evident that our algorithm yields favorable results across all the databases, demonstrating the effectiveness of our approach in optimizing the hyperparameters of MLPNNs. By achieving good results across various databases, we can infer that our algorithm excels in optimizing crucial hyperparameters of MLPNNs. These hyperparameters include the number of neurons in the hidden layers, the number of hidden layers, and the choice of activation function. Optimizing these hyperparameters is essential for enhancing the performance and predictive capabilities of neural networks.

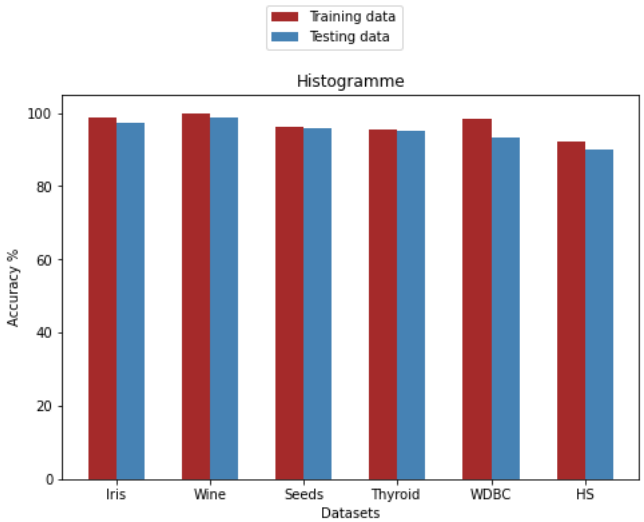

Figure 4. Comparison between classification rates of training and testing data.

As shown in the figure 4, the classification rate comparison results of training data and test data show that our method is a good model to solve the generalization problem. We see that, across all datasets, the two rates are very close to each other. This shows that our method works well when it comes to solving issues with generalized learning.

5.3. Results comparison

In this section, we will compare our approach with popular hyperparameter optimization methods, then with other machine learning algorithms.

5.3.1. Compared to other hyperparameter optimization methods In this partie, we propose to compare the results obtained with different methods on four databases, of which those in table 1. There are many hyperparameter optimization (HPO) method in the literature; we describe some of them, and we get the comparison results with our proposed method.

- Grid search (GR) [17] is an optimizing method that makes it possible for us to test a set of parameters and then compare the performance to determine the best setting.
- Random Search (RS) [18] in the search space, RS randomly selects a pre-defined number of samples between the upper and lower bounds as candidate hyper-parameter values, and then trains these candidates until the defined budget is exhausted.
- Successive Halving (SH) [20] is essentially an optimization algorithm, which randomly samples a set of hyperparameter configurations; it evaluates the performance of all currently remaining configurations; it ignores, the bottom half of the lowest scoring configurations repeats until only one configuration remains.
- Hyperband is an extension on top of successive halving algorithms, proposed by Lisha Li and others in [21].

Hyperparameter search space for a MLPNN. Keep in mind that the actual search space can vary depending on the specific problem and requirements. In this experiment we will use the intervals shown in Table 3.

Table 3. Search space of Hyper-parameter.

Hyper-parameter	Type	Search space
Number of neurons	Discrete	$\{1, \ldots, 20\}$
Number of layers	Discrete	$\{1, \ldots, 4\}$
Learning Rate μ	Continue	$[0.02, 0.2]$
Activation Function	Nominal	Sigmoid, ReLU, SoftPlus, Tanh
Number of Epochs	Discrete	{50,100,150,200}

Table 4 shows the performance of the hyperparameter optimization methods in terms of precision and execution time in seconds TC(s).

Based on the given information in table 4, it seems that the default HP configurations used in the experiment did not produce the optimal model performance. This underscores the importance of employing hyperparameter optimization methods to enhance model performance. Table 4 and figure 5 demonstrates that genetic grid search and random search often necessitate significantly more computation time compared to other optimization methods, despite achieving satisfactory accuracy. Conversely algorithm GA, Hyperband and Successive Halving are quicker but yield slightly lower accuracy compared to other methods. Our approach exhibits the best model performance within a reasonable timeframe. It guarantees the identification of the most suitable hyperparameters and parameters for MLPNN models. Therefore, the hyperparameter optimization (HPO) approach employed in the experiment surpasses GS, RS, Hyperband, Successive Halving, GA in terms of both accuracy and computation time.

5.3.2. Compared to other machine learning algorithms In this party we will compared our method with various established machine learning models commonly used in classification tasks:

Table 4. Performance evaluation of applying HPO methods to MLPNN.

Dataset	Method	Accuracy	TC(s)
Iris	Default HPs	96.00	1.02
	Grid Search	97.33	15.64
	Random Search	96.00	8.12
	Successive halving	96.00	6.81
	Hyperband	97.33	4.23
	Genetic Algorithm	96.00	3.21
	P.method	**97.33**	**2.59**
Wine	Default HPs	96.22	1.17
	Grid Search	98.88	20.98
	Random Search	97.00	7.14
	Successive halving	97.88	5.62
	Hyperband	98.56	4.12
	Genetic Algorithm	97.82	4.01
	P.method	**98.88**	**3.59**
Seeds	Default HPs	90.32	1.15
	Grid Search	95.00	19.12
	Random Search	93.25	10.20
	Successive halving	92.23	9.42
	Hyperband	93.50	6.65
	Genetic Algorithm	93.00	5.02
	P.method	**95.95**	**3.55**
WDBC	Default HPs	94.00	3.14
	Grid Search	98.00	47.54
	Random Search	97.25	24.73
	Successive halving	98.00	17.00
	Hyperband	98.12	12.23
	Genetic Algorithm	97.21	13.21
	P.method	**98.48**	**9.42**

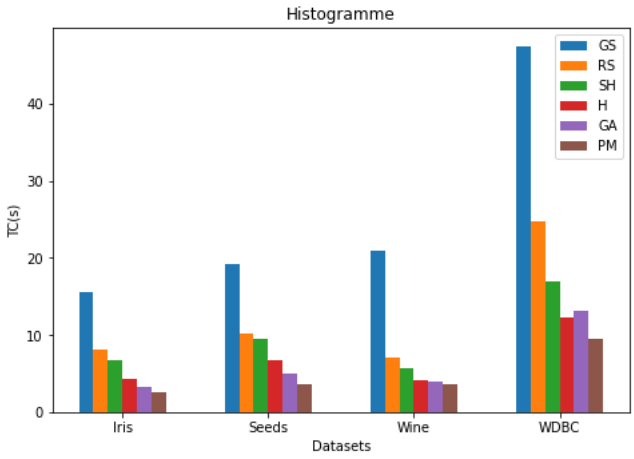

Figure 5. Comparison of execution times between different HPO methods.

- K-Nearest Neighbors (KNN): A straightforward algorithm for classification and regression tasks. It assigns a class to a data point based on the majority class among its k nearest neighbors in the feature space, making no assumptions about the data distribution.
- Support Vector Classifier (SVC): Also known as Support Vector Machine (SVM), it identifies the hyperplane that optimally separates classes in the feature space. SVC accommodates linear and non-linear decision boundaries via various kernel functions.
- Decision Tree Classifier (DT): Adopts a tree-like structure, with each internal node representing a feature and each leaf node denoting a class label. Known for interpretability and applicability across domains, decision trees capture non-linear relationships.
- Random Forest Classifier (RF): Based on ensemble learning, it creates numerous decision trees during training and determines predictions by selecting the most frequently occurring class among these trees. Random Forest reduces overfitting risk by training each tree on a random subset of data and features.
- Gradient Boosting Classifier (GB): Sequentially builds multiple weak learners, with each correcting errors made by its predecessors. Decision trees are typically employed as weak learners, enhancing predictive performance through iterative minimization of a loss function.
- AdaBoost Classifier (AB): Combines multiple weak learners to form a strong classifier by assigning higher weights to incorrectly classified instances, iteratively adjusting instance weights to enhance classification accuracy.
- Extreme Gradient Boosting Classifier (XGB): An advanced implementation of gradient boosting prioritizing speed and performance. It offers numerous hyperparameters for fine-tuning performance and is widely used in data science competitions and production environments.

The table 5 presents the performance metrics of various classifiers, encompassing KNeighbors, SVM, Decision Tree, RandomForest, Gradient Boosting, AdaBoost, XGB, and a proposed method.

Table 5. Email Spam Classification Results of Different Algorithms

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1-score**
Random Forest	0.94	0.95	0.90	0.93
Naive Bayes	0.81	0.69	0.96	0.80
SVM	0.92	0.93	0.87	0.90
Gradient Boosting	0.95	0.96	0.91	0.94
KNN	0.91	0.94	0.84	0.89
AdaBoost	0.94	0.94	0.90	0.92
XGBoost	0.95	0.96	0.92	0.94
P.method	**0.96**	**0.97**	**0.92**	**0.94**

The table 5 provides a comprehensive overview of the performance metrics of various algorithms used in email spam classification. Each algorithm is evaluated based on its accuracy, precision, recall, and F1 score. Notably, ensemble methods such as Random Forest, Gradient Boosting, AdaBoost, and XGBoost demonstrate consistently high performance across all metrics, demonstrating their effectiveness in accurately identifying spam emails. Conversely, Naive Bayes exhibits high recall but relatively low precision, potentially indicating a tendency to classify non-spam emails as spam. Support Vector Machine (SVM) provides balanced results on all metrics, while K-Nearest Neighbors (KNN) achieves decent accuracy and precision but shows a lower recall rate.

However, the most efficient among the algorithms is the proposed method (called P.method). This method integrates a sophisticated approach to selection of activation functions and optimization of the neural network structure, formulated by mixed variable optimization. By combining stochastic optimization with backpropagation algorithm, the P. method obtains remarkable results with precision of 0.96, precision of 0.97, recall of 0.92 and F1 score of 0.94 . What sets the P. method apart is its ability to outperform even ensemble methods while maintaining computational efficiency. The hybrid nature of the algorithm allows it to strike a balance between complexity and execution time, making it a promising solution for spam classification tasks. Overall, the P. method

presents a compelling advancement in the field, offering superior performance and efficiency compared to existing approaches.

Moreover, the versatility of our proposed model and algorithm extends across various domains reliant on deep learning. These include enhancing computer vision tasks such as object detection and image classification, improving medical imaging analysis and patient diagnosis in healthcare, and refining financial forecasting and risk management tools. The integration of our approach into existing workflows involves incorporating the model into training pipelines and leveraging automated hyperparameter tuning techniques for architecture optimization. This integration promises streamlined model development and deployment, ultimately leading to more efficient and effective deep learning solutions across diverse real-world applications.

6. Conclusion

In this paper, we have introduced a combinatorial optimization model for multilayer perceptron to architecture optimization and activation functions selection. The main goal of our model is to determine the ideal number of neurons, hidden layers and and activation functions selection in an MLPNN, ensuring that the resulting neural network is capable of solving problems effectively without succumbing to the issues of overfitting or underfitting. By achieving this, our contribution enhances the generalization capacity of MLPNNs. One of the key aspects of our approach is the proposed algorithm used to approximate the solution of the model. This algorithm has demonstrated its effectiveness through the results obtained in our experiments. The solutions obtained using our algorithm showcase its ability to find good solutions, further reinforcing the utility and value of our approach. Looking ahead, our future endeavors will focus on delving into the selection of hyperparameter spaces. This exploration promises to shed light on refining our methodology further, thereby bolstering its applicability across diverse domains. Moreover, we envision extending the application of our methodology to real-life scenarios, thereby broadening its scope and utility in addressing practical challenges beyond the confines of theoretical frameworks.

REFERENCES

1. Ehteram, M., Ahmed, A. N., Khozani, Z. S., & El-Shafie, A. (2023). Graph convolutional network–Long short term memory neural network-multi layer perceptron-Gaussian progress regression model: A new deep learning model for predicting ozone concertation. Atmospheric Pollution Research, 14(6), 101766.
2. Xie, Y., Ueda, Y., & Sugiyama, M. (2021). A two-stage short-term load forecasting method using long short-term memory and multilayer perceptron. Energies, 14(18), 5873.
3. Desai, M., & Shah, M. (2021). An anatomization on breast cancer detection and diagnosis employing multi-layer perceptron neural network (MLP) and Convolutional neural network (CNN). Clinical eHealth, 4, 1-11.
4. Ghimire, S., Nguyen-Huy, T., Prasad, R., Deo, R. C., Casillas-Perez, D., Salcedo-Sanz, S., & Bhandari, B. (2023). Hybrid convolutional neural network-multilayer perceptron model for solar radiation prediction. Cognitive Computation, 15(2), 645-671.
5. Yu, X., Efe, M. O., & Kaynak, O. (2002). A general backpropagation algorithm for feedforward neural networks learning. IEEE Transactions on Neural Networks, 13(1), 251–254.
6. Han, F., Jiang, J., Ling, Q.-H., & Su, B.-Y. (2019). A survey on metaheuristic optimization for random single-hidden layer feedforward neural network. Neurocomputing, 335, 261–273.
7. Moshkbar-Bakhshayesh, K. (2020). Performance study of bayesian regularization based multilayer feed-forward neural network for estimation of the uranium price in comparison with the different supervised learning algorithms. Progress in Nuclear Energy, 127, 103439.
8. Ramchoun, H., & Ettaouil, M. (2020). New prior distribution for Bayesian neural network and learning via Hamiltonian Monte Carlo. Evolving Systems, 11(4), 661–671.
9. Carvalho, A. R., Ramos, F. M., & Chaves, A. A. (2011). Metaheuristics for the feedforward artificial neural network (ANN) architecture optimization problem. Neural Computing and Applications, 20(8), 1273–1284.
10. Ramchoun, H., Ghanou, Y., Ettaouil, M., & Janati Idrissi, M. A. (2016). Multilayer perceptron: Architecture optimization and training.
11. Li, Q., Yan, M., & Xu, J. (2021). Optimizing convolutional neural network performance by mitigating underfitting and overfitting. 2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS), 126–131. IEEE.
12. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., & Leyton-Brown, K. (2013, December). Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In NIPS workshop on Bayesian Optimization in Theory and Practice (Vol. 10, No. 3).
13. Ali, Y. A., Awwad, E. M., Al-Razgan, M., & Maarouf, A. (2023). Hyperparameter search for machine learning algorithms for optimizing the computational complexity. Processes, 11(2), 349.

14. El-Hassani, F. Z., Amri, M., Joudar, N. E., & Haddouch, K. (2024). A New Optimization Model for MLP Hyperparameter Tuning: Modeling and Resolution by Real-Coded Genetic Algorithm. Neural Processing Letters, 56(2), 1-31.
15. Kuhn, M., Johnson, K., & Others. (2013). Applied predictive modeling (Vol. 26). Springer.
16. Abreu, S. (2019). Automated architecture design for deep neural networks. arXiv Preprint arXiv:1908. 10714.
17. Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. Advances in Neural Information Processing Systems, 24.
18. Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(2).
19. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., … Others. (2013). Towards an empirical foundation for assessing bayesian optimization of hyperparameters. NIPS Workshop on Bayesian Optimization in Theory and Practice, 10.
20. Karnin, Z., Koren, T., & Somekh, O. (2013). Almost optimal exploration in multi-armed bandits. International Conference on Machine Learning, 1238–1246. PMLR.
21. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research, 18(1), 6765–6816.
22. Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik, 4(1), 238–252.
23. Feng, X., Zhao, J., & Kita, E. (2021). Genetic algorithm-based optimization of deep neural network ensemble. The Review of Socionetwork Strategies, 15, 27–47.
24. Sreedharan, M., Khedr, A. M., & El Bannany, M. (2020). A multi-layer perceptron approach to financial distress prediction with genetic algorithm. Automatic Control and Computer Sciences, 54, 475–482.
25. Rajpoot, A., Saluja, G., Malsa, N., & Gupta, V. (2022). A Particle Swarm Optimization Based ANN Predictive Model for Statistical Detection of COVID-19. Artificial Intelligence in Healthcare, 21–34.
26. Chatterjee, S., Sarkar, S., Hore, S., Dey, N., Ashour, A. S., & Balas, V. E. (2017). Particle swarm optimization trained neural network for structural failure prediction of multistoried RC buildings. Neural Computing and Applications, 28, 2005–2016.
27. Hamed, H. N. A., & Haza, N. (2006). Particle swarm optimization for neural network learning enhancement. Universiti Teknologi Malaysia.
28. Gupta, T. K., & Raza, K. (2020). Optimizing deep feedforward neural network architecture: A tabu search based approach. Neural Processing Letters, 51, 2855–2870.
29. Heidari, A. A., Faris, H., Mirjalili, S., Aljarah, I., & Mafarja, M. (2020). Ant lion optimizer: theory, literature review, and application in multi-layer perceptron neural networks. Nature-Inspired Optimizers: Theories, Literature Reviews and Applications, 23–46.
30. Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., … Others. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. Information Fusion, 76, 243–297.
31. Wythoff, B. J. (1993). Backpropagation neural networks: a tutorial. Chemometrics and Intelligent Laboratory Systems, 18(2), 115–155.
32. Du, S., Lee, J., Li, H., Wang, L., & Zhai, X. (2019). Gradient descent finds global minima of deep neural networks. International Conference on Machine Learning, 1675–1685. PMLR.
33. Asuncion, A., & Newman, D. (2007). UCI machine learning repository.