Neural Processing Letters (2024) 56:105
https://doi.org/10.1007/s11063-024-11578-0

A New Optimization Model for MLP Hyperparameter Tuning: Modeling and Resolution by Real-Coded Genetic Algorithm

Fatima Zahrae El-Hassani[1] · Meryem Amri[1] · Nour-Eddine Joudar[2] · Khalid Haddouch[1]

Accepted: 16 February 2024

Abstract
This paper introduces an efficient real-coded genetic algorithm (RCGA) evolved for constrained real-parameter optimization. This novel RCGA incorporates three specially crafted evolutionary operators: Tournament Selection (RS) with elitism, Simulated Binary Crossover (SBX), and Polynomial Mutation (PM). The application of this RCGA is directed toward optimizing the MLPRGA+5 model. This model is designed to configure Multilayer Perceptron neural networks by optimizing both their architecture and associated hyperparameters, including learning rates, activation functions, and regularization hyperparameters. The objective function employed is the widely recognized learning loss function, commonly used for training neural networks. The integration of this objective function is supported by the introduction of new variables representing MLP hyperparameter values. Additionally, a set of constraints is thoughtfully designed to align with the structure of the Multilayer Perceptron (MLP) and its corresponding hyperparameters. The practicality and effectiveness of the MLPRGA+5 approach are demonstrated through extensive experimentation applied to four datasets from the UCI machine learning repository. The results highlight the remarkable performance of MLPRGA+5, characterized by both complexity reduction and accuracy improvement.

All authors have contributed equally to this work.

✉ Fatima Zahrae El-Hassani
fatimazahrae.elhassani@usmba.ac.ma

Meryem Amri
meryem.amri@usmba.ac.ma

Nour-Eddine Joudar
n.joudar@um5r.ac.ma

Khalid Haddouch
khalid.haddouch@usmba.ac.ma

[1] Engineering System, and Applications Laboratory, ENSA, Sidi Mohamed Ben Abdellah University, Fez, Morocco

[2] Modelling and Mathematical Structures Laboratory, Department of Mathematics, FST, Sidi Mohamed Ben Abdellah University, Fez, Morocco

Published online: 14 March 2024

Keywords Real-coded genetic algorithm · Hyperparameter · Constrained optimization · Multi-layer perceptron

1 Introduction

In recent years, the problem of supervised learning with multilayer perceptron neural networks (MLPNN) has received a lot of attention in the research community. Various approaches have been proposed to solve this problem, including classical learning [1], optimization methods [2], and their effectiveness has been widely discussed.

Developing an effective neural network is a complex and essential task in machine learning and artificial intelligence [3]. It involves the intricate selection of the network's architecture and training configurations to achieve optimal performance. These decisions are guided by a set of hyperparameters, which are additional parameters that play a crucial role in defining the network's behavior but cannot be learned directly from the training data [4]. Hyperparameters encompass a wide range of settings, including the number of hidden layers, learning rates, batch sizes, regularization parameters, activation functions, and more [5]. Properly tuning these hyperparameters is vital for achieving superior neural network performance [6]. The impact of hyperparameters on neural network performance is well-documented in the literature, and optimizing them is essential for achieving top-tier performance [3, 7, 8]. Selecting optimal values for hyperparameters is a critical aspect of developing a high-performing neural network, as it can significantly impact the network's accuracy, generalization ability, and training speed [9]. To address the challenge of hyperparameter optimization, several approaches have been proposed, including grid search [10], random search [11], Bayesian optimization [12], and genetic algorithms [13]. While some of these approaches have proven effective in certain cases, they can be computationally expensive and may not be suitable for optimizing real-valued hyperparameters and large-scale neural networks [11, 14, 15].

Due to its robust capabilities in tackling real-world optimization problems, the real-coded genetic algorithm (RCGA) has established itself as a highly effective and commonly utilized evolutionary algorithm (EA). Recent literature has presented numerous successful applications of RCGAs across diverse fields [16–20], underscoring its prominence and relevance in contemporary optimization practices. Drawing inspiration from the principles of natural selection and the survival of the fittest observed in the biological world, real-coded genetic algorithms (RCGAs) essentially operate as population-based stochastic search schemes. They incorporate a structured sequence of selection, crossover, and mutation operators to explore and optimize solutions. Over the past few decades, significant progress has been made to enhance the solution efficiency of RCGAs. These advancements and dedicated efforts have led to the categorization of previous developments and attempts based on the mechanisms and techniques employed. In a recent study conducted by Chuang and colleagues [21], a comprehensive survey was undertaken to explore the advancements in crossover operators [22–28]. Their research findings revealed that two predominant schemes, namely line segment connection and distribution analysis of parents, have been widely employed in the development of novel crossover operators. However, it was observed that these crossover operators face challenges when dealing with highly demanding optimization problems. Specifically, these operations may struggle to generate viable offspring in regions of ambiguity, especially in scenarios where the population size is relatively small in comparison to the overall search space or when the initial population distribution lacks uniformity across the permissible domain [21]. Furthermore, the designed crossover operators may encounter difficulties in locating

the global optimum, particularly when the true solution is in close proximity to or situated along the boundaries of the feasible search space [23]. To address the challenges associated with crossover operators, particularly in cases where optimization problems are subject to stringent conditions, this study introduces a novel approach that employs the Simulated Binary Crossover (SBX) operator [29]. SBX is designed to facilitate efficient exploration of the search space, even in the presence of complex constraints [30]. Unlike conventional crossover operators that rely on fixed or random step sizes for recombination, SBX takes an adaptive approach. It adjusts the step size dynamically based on the relative fitness information obtained from the objective function and the prevailing constraint violation status [31]. This adaptability allows SBX to effectively explore the continuous search space for real-valued parameters, enhancing its ability to efficiently address complex optimization tasks. In addition to the utilization of the Simulated Binary Crossover (SBX) operator, this paper also leverages the benefits of the polynomial mutation operator [32] to further enhance the optimization process. Polynomial mutation serves as a valuable complement to the crossover operator by introducing additional diversity and exploration capability in the population [33]. This mutation operator operates by perturbing individual gene values, thus enabling the algorithm to explore uncharted regions of the solution space.

The application of these genetic operators is tailored to the optimization of the MLPRGA+5 model, which serves as a comprehensive approach to configuring Multilayer Perceptron neural networks. By optimizing both the network's architecture and the associated hyperparameters, which encompass variables like learning rates, activation functions, and regularization hyperparameters, this algorithm efficiently explores complex optimization landscapes. This multi-faceted approach enhances the algorithm's capacity to fine-tune hyperparameters and optimize neural network architectures effectively, resulting in superior performance when applied to tasks such as classification.

To evaluate the proposed approach, the study uses four datasets from the UCI repository of machine learning databases [34]: "Iris," "Hypothyroid," "breast cancer," and "Wine". The MLPRGA+5 approach showcases its efficiency through numerical and manual analysis, demonstrating its exceptional performance in both complexity reduction and accuracy enhancement. The main objectives of this paper are given as follows:

- Provide a new optimization model for MLP Hyperparameters tuning.
- solve the proposed model using a real-coded genetic algorithm
- comparison of the proposed approach with existing MLP-based models

The paper is structured as follows: In Sect. 2, we recommended several existing papers that address the optimization of hyperparameters in neural networks and provided contextual information on various methods used for training MLPNNs after establishing the topic under research in Sect. 1. We will provide a new optimization Model for MLP Hyperparameters Tuning in Sect. 3, In addition, we provide a detailed description of the proposed model, including the argumentation for the objective function and the proposed constraints. Section 4 is dedicated to the Real-Coded Genetic Algorithm (RCGA) solver for MLP, which will be adapted to fit the proposed model MLPRGA+5 in Sect. 3. Finally, Sect. 5 outlines experiments and results before concluding in Sect. 6.

2 Related Works

Multilayer Perceptron (MLP) model weight and parameter tuning is a critical machine learning task as it directly impacts the model's performance. The architecture and hyperparameters

of MLPs have been proposed to be optimized in recent years utilizing a variety of optimization techniques, including gradient descent, random search, and Bayesian optimization. However, due to their effectiveness in handling high-dimensional and non-linear search spaces, genetic algorithms (GA) have attracted interest. In terms of model accuracy and generalization, the use of GA in optimizing MLP designs and hyperparameters has produced encouraging results. In this section of related work, we evaluate the most recent results in MLP model optimization using genetic algorithms and discuss the advantages and limitations.

The aim of MLPNN training is to get at an optimal objective function that best represents the weights selection, which can be thought of as an optimization problem. Once the optimization model has been created, a number of alternative algorithms can be applied. Traditional methods like back-propagation (BP) have previously been used for MLPNN training [35]. However, it has been shown that the optimization of MLP parameters and hyperparameters is effective when utilizing complex optimization methods such as genetic algorithms (GA). The advantage of GA is that it uses parallel search processes to investigate a large search space in search of the optimum solution, reducing the possibility of getting stuck at a local minimum. The use of genetic algorithms to improve the weights of MLP models has been the subject of numerous studies [36]. Wang et al. [37] use the GA approach to optimize the initial weights and biases of the back-propagation (BP) neural network in order to anticipate the bending force in the hot strip. Jasmeen Gill et al. [38] Use back-propagation (BP) and genetic algorithms (GA), for forecasting the weather the experiment uses daily weather data. Saremi et al. [39] improved the weights of an MLP model for speech emotion identification using a hybrid GA and BP, and she did it with greater accuracy than she could have with BP alone. MLP weight optimization has also been done using other optimization methods, such as a hybrid GA with simulated annealing (GA-SA) and a hybrid GA with PSO, both of which performed better than using GA or the other algorithm alone.

Optimizing architecture and hyperparameters of MLP models can also be seen as an optimization problem [3]. Multilayer-perceptron (MLP) neural networks' performance has been optimized in several research by changing various architecture parameters and hyperparameters. Architecture parameters, such as the number of hidden layers, the number of neurons per hidden layer, and the type of activation function, describe the structure of the MLP. Sam Ansari et. [40] utilized a genetic algorithm to find the best parameter values for a multi-layer perceptron utilizing different chromosomal coding techniques, while Jenny Domashova et al. [41] used it to identify the best architecture for precisely classifying situations. Sreedharan et al. [42]combines a Multi-Layer Perceptron (MLP) neural network with a Genetic Algorithm (GA) to predict financial distress in businesses. The GA optimizes hyperparameters like hidden layers, neuron counts, and activation functions, improving the MLP's performance. This hybrid approach offers an effective method for early financial distress prediction, aiding stakeholders in risk assessment decisions. Daviran et al. [43] focuses on landslide susceptibility prediction and employs Artificial Neural Networks (ANN). The unique aspect of the study is the use of a Genetic Optimization Algorithm to fine-tune the hyperparameters of these models. This study by Kumar et al. [44] focuses on optimizing the hyperparameters of Deep Neural Networks (DNN) through a two-step genetic approach. The research introduces a novel method to fine-tune hyperparameters, aiming to enhance the performance of DNNs. The twofold genetic approach involves the use of Genetic Algorithms (GA) to explore hyperparameter combinations and Genetic Programming (GP) to evolve neural architectures. This innovative approach provides a comprehensive and effective strategy for optimizing DNNs, ultimately improving their performance in various applications. Mohan et al. [45] introduces an innovative automated superlearner that combines multiple machine learning models for improved predictive accuracy. This superlearner is optimized using a Genetic Algorithm-

based hyperparameter tuning approach. By leveraging Genetic Algorithms, the study seeks to find the best hyperparameters for each model within the superlearner, leading to enhanced predictive performance. This research presents a novel method for automated model selection and hyperparameter optimization, contributing to more accurate predictions in various applications. Abdollahiet al. [46] focuses on predicting diabetes using a hybrid stacked ensemble approach. GAs are utilized to optimize the hyperparameters of the ensemble, resulting in a more accurate diabetes prediction model. The study showcases the effectiveness of combining ensemble methods with GA-based hyperparameter tuning for medical diagnosis tasks, particularly in the context of diabetes prediction. By optimizing hyperparameters, the research aims to enhance the accuracy of landslide susceptibility predictions. Hong Wang et al. [47] improved the architecture of an artificial neural network for computing slope stability safety factors. Tayebi et al. [48] analyzes the performance of metaheuristic algorithms in optimizing hyperparameters for fraud transaction detection models. It examines the efficacy of various metaheuristic techniques for improving the accuracy of fraud detection systems. The study offers valuable insights into the application of metaheuristics for hyperparameter optimization in the context of fraud prevention. This study employed by Arukonda et al. [49] introduces a novel ensemble approach for disease diagnosis that emphasizes diversity among models. It incorporates Genetic Algorithms (GAs) to enhance diversity and optimize the ensemble's performance, leading to effective disease diagnosis. The research focuses on improving diagnostic accuracy by harnessing the strengths of diverse models and GA-based hyperparameter tuning, offering a promising strategy for disease diagnosis. In order to assess indoor environmental conditions in real-time, Miguel Martinez-Comesa et al. [50] recommended employing optimized MLP neural networks. He used the multiobjective genetic algorithm NSGA-II to determine the architecture with the lowest error and complexity. Furthermore, by adding a decision variable to each layer, researchers have created a mathematical model to optimize hidden layers [2, 51]. The MLP model's learning parameters, such as learning rate, momentum, activation function, and regularization parameters, are referred to as hyperparameters. To optimize hyperparameters, many conventional methods have been applied, including grid search and random search. The hyperparameters of MLP models have been optimized using more sophisticated optimization methods, such as genetic algorithms, PSO, and Bayesian optimization, which have improved generalization and increased accuracy.

Li et al. [52], for example, introduced a hybrid GA and PSO algorithm to optimize the hyperparameters of MLP models for financial time series forecasting, outperforming grid search and random search in terms of performance. In comparison to grid search and random search, Wang et al. [53] use Bayesian optimization to optimize the hyperparameters of MLP models for breast cancer diagnosis producing greater accuracy. Additionally, Zhang et al. [54] used an adaptive GA to outperform various cutting-edge optimization strategies in optimizing the hyperparameters of MLP models for multi-label classification tasks. Particle swarm optimization (PSO) [55], differential evolution (DE) [56], and simulated annealing (SA) [57]are further optimization methods that have been used to MLP models.

3 A new Optimization Model for MLP Hyperparameter Tuning

Hyperparameter tuning task in MLP involves solving optimization problems. When constructing an MLP model, the weight parameters are initialized and iteratively optimized using various optimization methods until the objective function reaches a minimum value or the accuracy reaches a maximum value [3]. Similarly, hyperparameter optimization methods

focus on optimizing the model's architecture by identifying the most suitable hyperparameter configurations. In this section, we introduce the core principles of mathematical optimization and hyperparameter optimization, along with the proposed approach.

3.1 Hyperparameter Tuning in MLPS

In order to develop machine learning models that use hyperparameter optimization, it is necessary to explore the space of possible hyperparameter values to identify the parameters that make the model work best. By employing optimization techniques, we may successfully navigate this search space and find the hyperparameters that lead to the model operating at its most effective level. The four main components of hyperparameter optimization are an estimator with an objective function, a search space, an optimization method, and an evaluation function. These components work together to determine the best set of hyperparameters for the model.

In mathematics, the term "optimization" refers to the process of selecting the best course of action among a variety of feasible options in order to maximize or reduce the objective function [58]. The unconstrained optimization problem is indicated by [59]:

$$\min_{X \in \mathbb{R}} f(X) \tag{1}$$

Contrarily, constrained optimization can be expressed as [59]:

$$\begin{cases} \min f(X) \\ \text{Subject to:} \\ g_i(X) \leq 0, \quad \mathrm{i} = 1, \ldots, \mathrm{m} \\ \mathrm{h_i}(X) = 0, \quad \mathrm{j} = 1, \ldots, \mathrm{p} \end{cases} \tag{2}$$

where $f(X)$ is the problem objective function, X is the domain of X, $\mathrm{g_i}(X) \leq 0, \mathrm{i} = 1, \ldots, \mathrm{m}$, and $\mathrm{h_i}(X) = 0, j = 1, \ldots, \mathrm{p}$, are inequality and equality constrained functions, respectively. The feasible domain D of X is as follows:

$$\mathrm{D} = \{X \in \mathrm{X} \backslash \mathrm{g_i}(X) \leq 0,\ \mathrm{h_i}(X) = 0\} \tag{3}$$

The objective of a hyper-parameter optimization task is to get [60]:

$$X^* = \arg\min_{X \in \mathbb{R}} f(X) \tag{4}$$

The best prediction model f^* can be determined by using [61]:

$$f^* = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L\left(f\left(\mathrm{x_i}\right), \mathrm{y_i}\right) \tag{5}$$

where L is the cost function value for each sample, n is the number of training data points, $\mathrm{x_i}$ is the feature vector of the i-th instance, y_i is the associated output. In supervised learning algorithms, there are numerous distinct loss functions, such as the square of Euclidean distance, cross-entropy, information gain, etc. [61].

3.2 Proposed Approach

Machine learning models aim to optimize performance by finding the best values for parameters that affect the output. One key type of parameter is weight parameters, which are initially

set and then adjusted using an optimization technique until the model achieves a high level of accuracy or reaches a critical point in its objective function.

In order to create an effective machine-learning model, it is important to strike a balance between accuracy and complexity. This can be achieved by using hyperparameter optimization to find the optimal values for the model's hyperparameters. By optimizing hyperparameters, we can ensure that the model is both accurate and not overly complex, resulting in a model that performs well on both training and testing data. This work introduces a novel mathematical programming approach for optimizing neural networks, which can be seen as a mixed-constraint non-linear programming model. This model possesses a unique capability - it allows us to introduce new variables into the objective function. These variables serve as a mechanism for optimizing hyperparameters within a Multilayer Perceptron (MLP). Notably, these variables also act as a pruning technique. Their role is to systematically identify and eliminate any hidden layers within the MLP that may not be contributing effectively to the model's performance. Additionally, they facilitate adjustments to the model's hyperparameters, enabling fine-tuning.

The application of this approach holds significant promise. By eliminating unnecessary hidden layers and optimizing hyperparameters, we effectively reduce the model's complexity. This streamlined structure, in turn, leads to a notable reduction in the number of parameters (or "weight loss"), resulting in a more efficient and computationally lightweight model. The approach introduces certain constraints that facilitate communication between different levels of the neural network, aiding in the optimization process. These constraints are an integral part of the model, serving to guide the optimization towards achieving excellent results.

This paper presents a new optimization model, in order to:

- To optimize the architecture parameters of the neural network.
- Combining hyperparameter tuning with architecture optimization.

The following parts will provide an explanation of the suggested model.

3.2.1 Notation

In order to model the issue of neural hyperparameters optimization, we had to construct some notation:

- X : Set of inputs us explicative variable of a model.
- N: Number of hidden layers
- L : Total number of hyperparameters
- n_0: Number of neurons in the input layer
- n_i: Denotes the number of nodes in the layer i, for $i = 1 \dots N$.
- n_o: Number of neurons in output layer.
- n_{opt}: Optimal number of hidden layers
- h_i : The output of hidden layer, for $i = 1 \dots N$.
- f: Activation function
- O: Optimizer
- α : Learning rate
- λ : Regularisation parameter
- Y: The predicted output of the model
- d: The desired output
- F: Transfer function of ANN

- W: The weights of the network with:

$$W = \left(w_{k,j}^{i}\right)^{\mathrm{i}}_{\substack{0\le i\le N\\ 1\le k\le n_i\\ 1\le j\le n_{i+1}}} \quad , \quad w_{k,j}^{\mathrm{i}} \in \mathbb{R}$$

- H: Denotes the search space of hyperparameters (configuration space).
  $H_i \begin{cases} \mathrm{n_i} \text{ if } 1 \le i \le N \\ \{f, O, \alpha, \lambda\} \text{ if } N+1 \le i \le L \end{cases}$
- $\mathrm{c_{n_i}}$: Binary variable for $i = 1 \ldots N$, as:
  $\mathrm{c_{n_i}} \begin{cases} 1 \text{ if the number of randomly generated neurons } \mathrm{n_i} \\ \qquad \text{associated with layer i is greater than 0} \\ 0 \text{ otherwise} \end{cases}$

A multilayer perceptron performs a transformation of the input variables:

$$\mathrm{Y = F(X; W; H) = \left(y_1, y_2, \ldots, y_{n_{N+1}}\right)} \tag{6}$$

where $\mathrm{H} = (\mathrm{H_1}, \ldots, \mathrm{H_L}) \in \mathbb{R}_+^L$ and $\mathrm{C} = \left(\mathrm{c_{n_1}}, \ldots, \mathrm{c_{n_{N-1}}}\right) \in \{0, 1\}^{N-1}$ the result of the hidden layer h_1, is calculated by the following term and represented in Fig. 1

$$h_1 = \begin{pmatrix} h_1^1 \\ \vdots \\ \vdots \\ h_1^j \\ \vdots \\ \vdots \\ h_1 \end{pmatrix} = \begin{pmatrix} f\left(\sum_{\mathrm{k}=1}^{n_0} w_{\mathrm{k},1}^0 x_k\right) \\ \vdots \\ f\left(\sum_{\mathrm{k}=1}^{n_0} w_{\mathrm{k,j}}^0 x_k\right) \\ \vdots \\ f\left(\sum_{\mathrm{k}=1}^{n_0} w_{\mathrm{k,n_1}}^0 x_k\right) \end{pmatrix} \tag{7}$$

where $\left(\mathrm{x_1}, \mathrm{x_2} \ldots \mathrm{x_{n_0}}\right)$ are the inputs of neural networks.

The result of the hidden layer h_i, is calculated by the following term and represented in Fig. 2:

$$h_i = \begin{pmatrix} h_i^1 \\ \vdots \\ \vdots \\ h_i^j \\ \vdots \\ \vdots \\ h_i^{n_i} \end{pmatrix} = h_{i-1}\left(1 - \prod_{k=1}^{i} c_{n_k}\right) + \prod_{k=1}^{i} c_{n_k} \cdot \begin{pmatrix} f\left(\sum_{k=1}^{n_{i-1}} w_{k,1}^{i-1} h_{i-1}^k\right) \\ \vdots \\ \vdots \\ f\left(\sum_{k=1}^{n_{i-1}} w_{k,j}^{i-1} h_{i-1}^k\right) \\ \vdots \\ \vdots \\ f\left(\sum_{k=1}^{n_{i-1}} w_{k,n_i}^{i-1} h_{i-1}^k\right) \end{pmatrix} \tag{8}$$

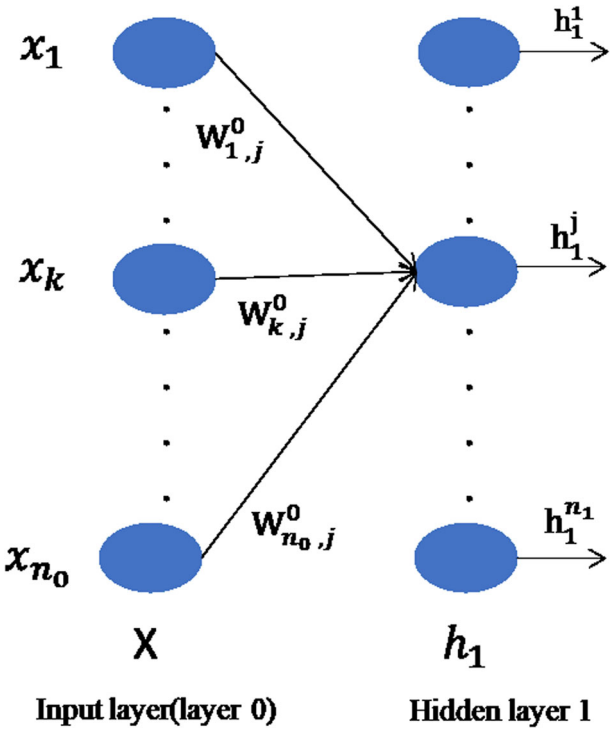

Fig. 1 Output of the first hidden layer

where i = 2 . . . N − 1 the result of the hidden layer h_N, is calculated by the following term and represented in Fig. 3

$$h_N = \begin{pmatrix} h_N^1 \\ \vdots \\ h_N^j \\ \vdots \\ h_N^{n_N} \end{pmatrix} = h_{N-1}\left(1 - \prod_{k=1}^{N} c_{n_k}\right) + \prod_{k=1}^{N} c_{n_k} \cdot \begin{pmatrix} f\left(\sum_{k=1}^{n_{N-1}} w_{k,1}^{N-1} h_{N-1}^k\right) \\ \vdots \\ \sum_{k=1}^{n_{N-1}} w_{k,j}^{N-1} h_{N-1}^k \\ \vdots \\ f\left(\sum_{k=1}^{n_{N-1}} w_{k,n_N}^{N-1} h_{N-1}^k\right) \end{pmatrix} \tag{9}$$

$$y_i = f\left(\sum_{k=1}^{n_N} w_{ki}^N h_N^k\right) \quad j = 1 \ldots n_{N+1} \tag{10}$$

With y_i as the output of the neural network

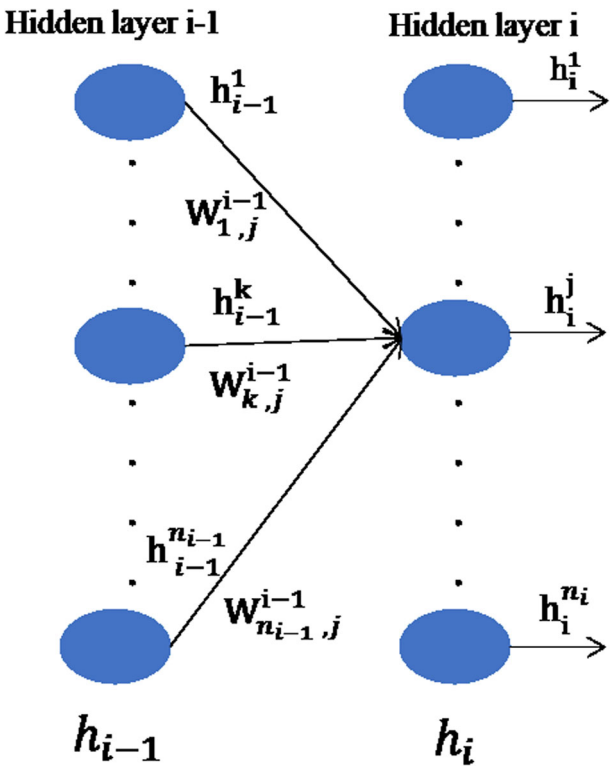

Fig. 2 Output of the ith hidden layer

3.2.2 Objective Function

The objective function utilized in the proposed model is the mean squared error (MSE), which is a widely-used error function for training neural networks. The MSE measures the average squared difference between the predicted output and the actual output. It is computed as the sum of the squared differences over all training examples in the example set:

$$Loss = \frac{1}{2J} \sum_{j=1}^{J} \left(F_j(X; W; H) - d_j\right)^2 \tag{11}$$

In the formula:

- J: represents the size of the example set, indicating the number of training examples.
- $F_j(X; W; H)$: denotes the predicted output based on the input X, the model weights W, and the hyperparameters H.
- d_j: represents the actual output for the corresponding training example.
- $\sum$: denotes the sum over all training examples $j = 1$ to J.

By calculating the squared differences between the predicted and actual outputs for each training example and summing them up, we obtain the MSE. This error function quantifies how well the model approximates the desired output for the given training examples.

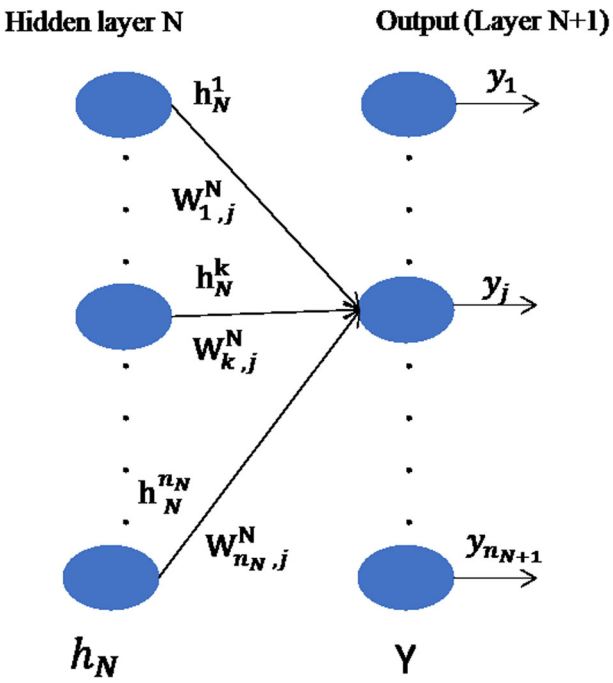

Fig. 3 Output of the ANN

The utilization of the MSE as the objective function allows the model to optimize its parameters and hyperparameters to minimize the discrepancy between the predicted and actual outputs. This optimization process aims to enhance the model's performance and improve its ability to generalize to unseen data.

3.2.3 Constraints and Proposed Model

To prevent the complete destruction of the hidden layer and ensure that there is at least one hidden layer, we can ensure that the first parameter is always greater than zero. This latter is stated as follows:

$$c_{n_1} = 1 \tag{12}$$

Setting the lower and upper boundaries of the hyperparameters' range of values is crucial to guaranteeing the algorithm produces valid results. This limits the range of numbers the

algorithm can consider and stops it from coming up with solutions outside of the acceptable range. We avoid unimportant portions of the search space and set the lower bound to the smallest value and the upper bound to the highest value.

- To define the range of float values for a hidden layer, we start by determining the minimum and maximum values it can take. Specifically, we assign the interval $[n_{i\,\min}, n_{i\,\max}]$ to the hidden layer.

$$\mathrm{n_{i\,min}} \leq \mathrm{n_i} \leq \mathrm{n_{imax}} \quad \forall \mathrm{i} = 2 \ldots \mathrm{N} \tag{13}$$

- For a given hyperparameter, such as $\mathrm{n_i}$, we can assign a valid range $[0, \mathrm{n_{i\,max}}]$ to define the values that can be optimized.

$$0 \leq \mathrm{n_i} \leq \mathrm{n_{i\,max}} \quad \forall \mathrm{i} = \mathrm{N}+1 \ldots \mathrm{L} \tag{14}$$

The concept of optimizing neural architecture can be represented by the following model:

$$\begin{cases} \mathrm{Min}\ \dfrac{1}{2J}\displaystyle\sum_{j=1}^{J}\left(F_j(X;W;H)-d_j\right)^2 \\ \text{Subject to :} \\ \mathrm{H} = (\mathrm{H_1}, \ldots, \mathrm{H_L}) \in \mathbb{R}^L \\ \mathrm{C} = \left(\mathrm{c_{n_1}}, \ldots, \mathrm{c_{n_{N-1}}}\right) \in \{0,1\}^{N-1} \\ \mathrm{c_{n_1}} = 1 \ \mathrm{Or} \ \displaystyle\sum_{i=2}^{N} \mathrm{c_{n_i}} \geq 1 \\ \mathrm{n_{i\,min}} \leq \mathrm{n_i} \leq \mathrm{n_{i\,max}} \quad \forall \mathrm{i} = 2 \ldots \mathrm{N} \\ 0 \leq \mathrm{n_i} \leq \mathrm{n_{i\,max}} \quad \forall \mathrm{i} = \mathrm{N}+1 \ldots \mathrm{L} \\ \mathrm{W} = \left(\mathrm{w^i_{k,j}}\right) \quad \begin{matrix} 1 \leq \mathrm{k} \leq \mathrm{n_i} \\ 1 \leq \mathrm{j} \leq \mathrm{n_{i+1}} \end{matrix} \quad \text{Where} \quad \mathrm{w^i_{kj}} \in \mathbb{R} \end{cases} \tag{15}$$

The optimal number of hidden layers:

$$c_{\mathrm{opt}} = \sum_{i=1}^{N-1} c_{n_i} \tag{16}$$

During the training phase, we employ a randomized and sequential data input strategy. This approach ensures that each data point enters the network without any predetermined order or bias, contributing to a comprehensive learning experience. However, the true strength of our approach lies in addressing two critical aspects: the determination of the optimal number of hidden layers and the configuration of weights. Deciding on the right number of hidden layers significantly influences the network's ability to model complex data relationships. Simultaneously, weight configurations govern the flow of information within the network, making them pivotal in fine-tuning network responses. Additionally, the choice of an appropriate training period is essential, as it dictates how long the neural network learns from data. The training duration should align with the network's complexity, enabling weights to adapt and optimize their responses according to the problem's intricacies. Our approach, therefore, encompasses both architectural considerations and temporal dynamics, with a focus on optimizing the neural network's potential.

Over time, various methods and strategies have been proposed to tackle the optimization of hyperparameters in Multilayer Perceptron (MLP) training. In this context, we advocate for

the utilization of Real-Parameter Genetic Algorithms (RGAs) to directly manage real-valued variables, as opposed to an initial conversion to binary strings, within the described model.

4 Real Coded GA for Proposed Approach: MLPRGA+5

Genetic algorithms (GAs) are search algorithms that are inspired by the principles of natural genetic populations to find solutions to a wide range of problems. Real-coded genetic algorithms (RCGAs) are a type of optimization algorithm that uses real-valued vectors to represent individuals in the population. However, in optimization problems involving continuous domain variables, it is more intuitive to represent genes directly as real numbers. This choice aligns the solutions closely with their natural formulation, effectively eliminating distinctions between the genotype (coding) and phenotype (search space). The utilization of real coding was initially introduced in specific applications, such as in the work of Lucasius et al. [62] for chemometric problems, Davis [63] for the utilization of meta-operators in finding optimal parameters for standard genetic algorithms, and in [64] for numerical optimization in continuous domains. For a comprehensive overview of real-coded genetic algorithms, one can refer to [65–67].

As mentioned earlier, real coding proves to be the most suitable encoding method for optimization problems involving parameters in continuous domains. Given our specific focus on optimizing hyperparameters for MLP neural networks, the utilization of real coding and its associated genetic operators becomes a logical choice. In our study, we have not employed binary (ordinary) genetic algorithms for this purpose. One of the primary advantages of using real-value encoding over binary encoding is the enhanced precision it offers. Binary coding, when applied to real-valued numbers, can lead to a loss of precision, particularly when a limited number of bits is used to represent each value [68].

Furthermore, real-value encoding results in notably shorter chromosome strings. This efficiency is particularly significant when optimizing hyperparameters for MLP networks. Our aim is to find an optimal set of hyperparameters that enables the MLP neural network to perform with high accuracy and minimal complexity. In this context, a chromosome or genotype encompasses all the hyperparameters, with each gene representing an individual hyperparameter value.

Below, we present the chromosome representations along with the developed selection, crossover, and mutation operators tailored to this encoding.

4.1 MLPRGA+5:Representation and Initialisation

An individual or chromosome is a vector of genes represented by floating point values. A choice variable is represented by each gene. From a predetermined bounded solution space, each chromosome indicates a randomly generated response (feasible region). The MLPRGA+5 optimization process involves two key sets of decision variables: the model structure, which is a major determinant of performance and can take on values between $n_{i\,min}$ and $n_{i\,max}$, and the model hyperparameters, which also influence performance and can range between n_{j} min and $n_{j\,min}$. A typical chromosome structure for optimizing model accuracy and cost under constraints is illustrated in Fig. 4.

The initial population is generated at random, the weights are given random values between [0, 1], and it consists entirely of the individuals who will be subjected to the genetic operators

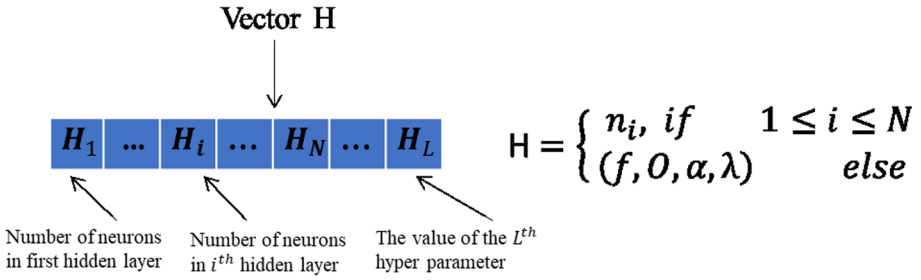

Fig. 4 Chromosome structure

and evaluated by the fitness function. After the initial population is created, each individual is evaluated and assigned a fitness value based on the fitness function.

4.2 MLPRGA+5:Fitness and Selection

An individual's extent of extemal adaptation can be determined using a statistic that is produced by a fitness function. This statistic aims to concentrate on the search for characteristics that will make an individual more adaptable or one who performs a position more effectively. The fitness in our study is indicated by the classification task accuracy rate. The percentage of all instances that were properly classified is known as accuracy. The fitness calculation method we propose is as follows:

$$\mathrm{F(i)} = \text{ Fitness (i)} = \text{ Accuracy (i)} \tag{17}$$

$$\text{Accuracy } = \frac{N_T}{N_d} \tag{18}$$

Where (N_T) and (N_d) stand for, respectively, the number of examples that were successfully classified and the total number of examples in a dataset.

In this study, we employ the two-individual tournament selection technique, as shown in Fig. 5. Two individuals are selected at random from the community, and the fitness function values are then contrasted to determine which of them has the highest value. This procedure was repeated for a second random pick of two individuals, and a winner was also chosen. The cross-over operator receives these two selected victors after that. We continue to use the elitist approach, which ensures that the best individuals are always passed on to the next generation unchanged.

4.3 MLPRGA+5:Reproduction Operators

Crossover is carried out using simulated binary crossover [29]. The polynomial mutation is used for mutation [32]. The following is a discussion of both of these procedures:

Simulated binary crossover: a genetic operator that creates two offspring solutions from two-parent solutions by simulating the binary crossover observed in nature. The procedure involves three steps:

Step 1:a uniform random number is chosen $U \in (0.1)$

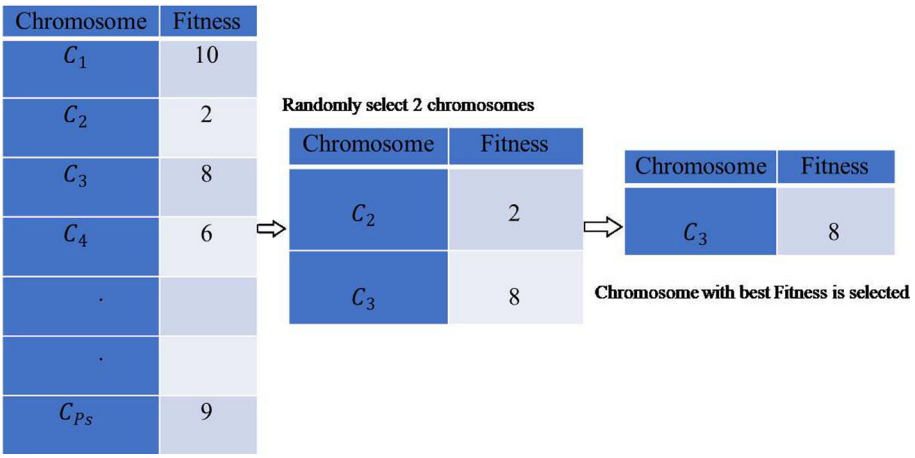

Fig. 5 Tournament selection

Step 2:a spreading factor β_q is calculated using the following equation:

$$\beta_q = \begin{cases} (2U)^{\frac{1}{\eta+1}} & \text{if } U \leq 0.5, \\ \left(\frac{1}{(2(1-U))}\right)^{\frac{1}{\eta+1}} & \text{othrwise} \end{cases} \tag{19}$$

The ordinate β_q is discovered so that the probability curve's area below it from 0 to β_q equals the selected random integer U. A single-point crossover in binary-coded GAs is found to have a similar search power to the probability distribution used to produce a child solution:

$$p(\beta) = \begin{cases} 0.5(\eta+1)\beta^n & \text{if } \beta \leq 1, \\ 0.5(\eta+1)\beta^{\frac{1}{\eta+2}} & \text{othrwise} \end{cases} \tag{20}$$

The uniform random number in the simulated binary crossover procedure is denoted by ' η ' and serves as the distribution index. This index is a non-negative real number that influences the probability of creating near-parent solutions. When the value of ' η ' is large, the probability of creating solutions that are similar to the parents is higher. In contrast, when ' η ' has smaller values, the probability of creating solutions that are distant from the parents is higher, resulting in a greater exploration of the search space.

Step 3: from the parent solutions $x_i^{(1,t)}$, $x_i^{(2,t)}$, two children's solutions, $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ can then be calculated using Equations (25) and (26).

$$\begin{aligned} x_i^{(1,t+1)} &= 0.5\left[\left(1+\beta_q\right)x_i^{(1,t)} + \left(1-\beta_q\right)x_i^{(2,t)}\right] \\ x_i^{(1,t+1)} &= 0.5\left[\left(1-\beta_q\right)x_i^{(1,t)} + \left(1+\beta_q\right)x_i^{(2,t)}\right] \end{aligned} \tag{21}$$

Polynomial Mutation: The objective of the mutation operation is to reestablish unexpected genetic individuals in order to prevent finding locally optimal solutions and thereby improve the population diversity and exploratory potential. What it does is as follows:

$$p' = \begin{cases} p + \overline{\delta_L}\left(p - x_i^{(L)}\right) & \text{for } U \leq 0.5 \\ p + \overline{\delta_R}\left(x_i^{(R)} - p\right) & \text{for } U > 0.5 \end{cases} \tag{22}$$

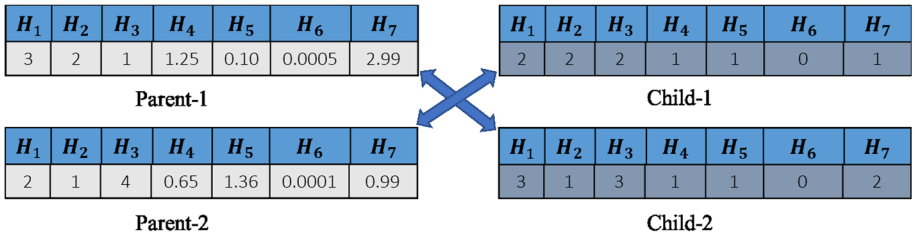

Fig. 6 Simulated binary crossover

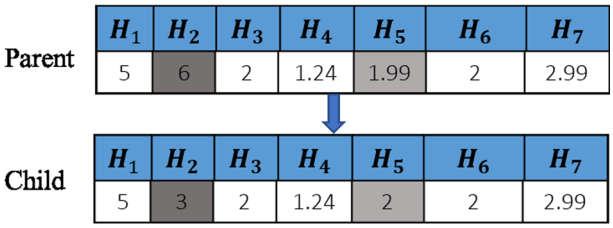

Fig. 7 Polynomial mutation

Where p′ is the child solution, p is the parent solution, $x_i^{(L)}$ and $x_i^{(R)}$ are the upper and lower limits of the parent component, respectively, the parameters $\left(\overline{\delta_L}, \overline{\delta_R}\right)$ are functions of U and η as shown in the following formula and they are calculated, as follows:

$$\begin{aligned} \overline{\delta_L} &= (2U)^{1/(1+\eta)} - 1 \qquad \text{for } U \leq 0.5 \\ \overline{\delta_R} &= 1 - (2(1-U))^{1/(1+\eta)}, \quad \text{for } U > 0.5 \end{aligned} \tag{23}$$

Here, η is the mutation distribution index and U is a uniform random integer between (0, 1), U is a uniform random integer between (0, 1). Figures 6 and 7 illustrate how the Simulated binary crossover and polynomial mutation functions respectively.

In this research, we harness the power of real-coded genetic algorithms (GAs) to optimize hyperparameters within the context of neural network configuration. The choice of real-coded GAs is underpinned by their distinct advantages, which make them a compelling choice for this intricate task. Unlike binary encoding or other discretization methods, real-coded GAs offer a seamless representation of the real-valued hyperparameters inherent in neural network architecture.

In conjunction with real-coded GAs, we employ a sophisticated selection method known as two-individual tournament selection. This technique introduces competition among individuals, enhancing the evolutionary process. Two individuals are randomly selected from the population, and their fitness function values are compared to determine the one with the highest fitness. This selection procedure is repeated for a second pair of randomly chosen individuals, and a winner is again selected. The chosen victors are then passed to the crossover operator. The utilization of tournament selection adds an element of diversity and competition to the selection process, allowing the algorithm to efficiently explore the solution space and maintain genetic diversity. Additionally, we integrate an elitist approach into our selection process, ensuring that the best individuals are always preserved and carried forward unchanged, which is critical in preventing the loss of highly fit solutions.

Complementing the power of real-coded GAs and tournament selection, we employ sophisticated genetic operators such as Simulated Binary Crossover (SBX) and Polynomial

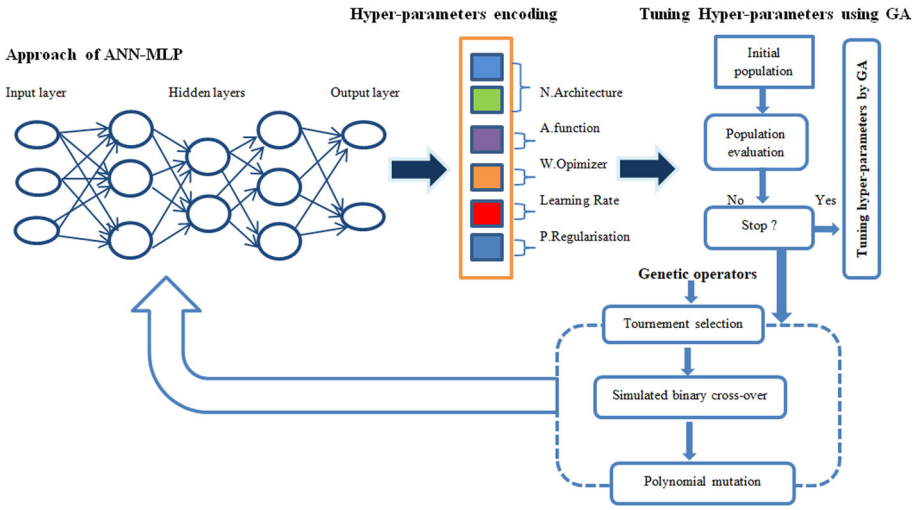

Fig. 8 The diagram shows the optimization of MLP hyper-parameters using RCGA. The hyper-parameters for neural architecture and activation function are encoded as a chromosome vector to be optimized by GA. Additionally, weight solver, regularization, and learning hyper-parameters are also optimized simultaneously

Mutation to further enhance the optimization process. SBX, a powerful crossover operator, is designed to balance exploration and exploitation. By simulating binary-like crossover for real-valued variables, SBX effectively maintains diversity in the population, thereby preventing premature convergence and efficiently exploring the search space.

In addition to SBX, we utilize Polynomial Mutation to introduce random perturbations to variable values. This mutation operator adds essential diversity to the population, mitigating the risk of falling into local optima and ensuring a comprehensive exploration of the solution space. The selection of real-coded GAs, SBX, and Polynomial Mutation is underpinned by their unique qualities, making them an ideal choice for classification tasks. Their adaptability and flexibility empower the simultaneous optimization of multiple hyperparameters within a unified framework, offering a comprehensive approach to hyperparameter tuning. This adaptability is vital in addressing the intricate interdependencies between hyperparameters in neural network configurations. Furthermore, the ability to strike a balance between exploration and exploitation efficiently guides the search process through the hyperparameter space, yielding optimized configurations tailored to classification tasks. Figure 8 presents the general flowchart methodology for the proposed process.

5 Experiments

The results of the model put out in section 4 will be reported in this part. Google Colab and TensorFlow 2.0+, which includes the Keras deep learning framework; the most recent version of sci-kit-learn, Numpy, and Deap were used to implement the model. The k-fold cross-validation is technic is utilized for the experiments that present the suggested hyperparameters optimization model. The provided results for each fold were generated using various test sets for each fold, which were not used for classifier training. The standard RCGA Algorithm 1

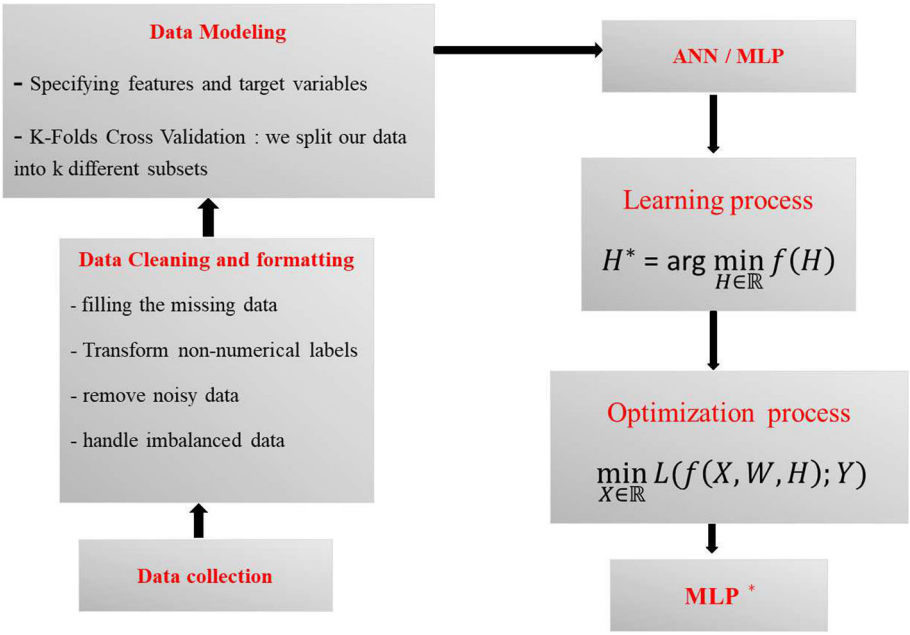

Fig. 9 The crucial steps in optimizing hyperparameters

used to optimize the hyperparameters of MLP neural network is as follows using the flow chart shown in Fig. 9.

Algorithm 1 Hyperparameter Optimization using Real-Coded Genetic Algorithm

Require:
P[0] : Initial population of hyperparameters generated randomly within defined bounds
pMLP: Original Multilayer Perceptron Neural Network
Number of generations
Termination condition
Ensure: Optimized model MLPGA+5 with tuned hyperparameters
Initialize P[0] with a random population of hyperparameters within defined bounds;
Initialize a new MLP with the same architecture as pMLP;
Initialize Generation i = 1;
for *i* from 1 to Number of Generation **do**
    Set hyperparameters of the MLP in P[iGeneration] with those from pMLP;
    Calculate Fitness [i] = Accuracy [i] in classification tasks based on the performance of MLP with hyperparameters in P[*i*];
    Perform Tournament Selection, Simulated Binary Crossover, and Polynomial Mutation to generate NewP;
    Increment Generation i by 1 ;
    Set *P*[iGeneration] = NewP;
end for
Compute the MLP with the optimal hyperparameters found in the last generation.

Table 1 Description of the used databases

Database	Features	Instances	Classes
Wine	13	178	3
Hypothyroid	24	3163	2
Iris	4	150	3
Cancer	9	699	2

5.1 Data Description and Preprocecing

A series of tests were carried out utilizing well-known benchmark datasets from the University of California Irvine (UCI) machine learning repository [69] in order to assess the performance of our suggested methodology. We specifically used four classification problems: the hypothyroid dataset, the iris dataset, the Wisconsin Breast Cancer dataset, and the wine dataset created by Fisher.

Table1. below provides a summary of the datasets utilized in this work, including details on each dataset's number of samples, properties, and target classes.

Wine The dataset includes 178 examples with three target classes and 13 real-valued characteristics. The outcomes of chemical examinations of wines produced in the same Italian region but from three different cultivars correlate to these classes, which correspond to 59, 71, and 48 examples, respectively. 13 distinct compounds found in each type of wine were quantified using chemical analysis.

Hypothyroid The Gravel Institute in Sydney, Australia provided the dataset for this study, which was then submitted to UC Irvine's Discovery in Databases. This dataset is a portion of a broader collection of datasets; however, the "hypothyroidism" dataset, which includes 3163 occurrences and 24 hypothyroidism-related variables, is the subject of this study. There are numerous missing values in the dataset.

Iris 50 samples of each of the three target classes-Iris Setosa, Iris Virginica, and Iris Versicolor-make up the 150 iris flower samples in the collection. Sepal length, sepal width, petal length, and petal width are the four real-valued properties present in each sample, and they offer information on the morphological traits of each iris species.

Breast Cancer Wisconsin dataset consists of 699 cases, 458 of which have been classified as benign and 241 as malignant. The goal is to categorize the breast tumor as benign or malignant, and the dataset has no missing variables. The issue is a two-class classification task, and it is based on several properties of the cells that make up the nuclei. While the remaining 350 patterns serve as a test set, the first 349 serve as a training set for optimizing the neural network weights.

We used hypothyroid and breast cancer data preparation approaches to ensure the correctness and dependability of our findings. Data preprocessing includes converting raw data into a practical format as well as carrying out cleaning and formatting procedures to get rid of any discrepancies or flaws that can influence our results. We handled unbalanced data, where there were significantly more negative samples than positive samples, by filling in an attribute means, converting non-numerical labels to numerical ones, reducing noisy data, and handling missing data (as indicated in the table). To maintain uniformity in the data across all datasets, we also standardized the data using a similar formula for each attribute individually.

Scaled Value $= \frac{X-\bar{u}}{S}$

where X: a sample, $\bar{u}$: mean of training samples for a single feature, S: a standard deviation of training samples for a single feature.

Table 2 Hyper-parameters to be optimized by GA, the default parameters of the MLP classifier, and the range of permitted values

Hyperparameter	Default value	Range of permitted values	gene
No of hidden layers	–	[1, 4]	–
No of neurons in hidden layer 1	–	[10, 15]	1
No of neurons in hidden layer 2	–	[−5, 10]	2
No of neurons in hidden layer 3	–	[−10, 10]	3
No of neurons in hidden layer 4	–	[−20, 10]	4
Activation	1-relu	1-tanh	
		2-relu	5
		3-logistic	
solver	2-adam	1-sgd	
		2-adam	6
		3-lbfgs	
Lambda	0.0001	[0.0001,2.0]	7
Alpha	1-constant	1 -constant	
		2-invscaling	8
		3-adaptive	

5.2 Parameters Setting of MLP and RCGAs

For classification tasks in our work, we used a Multilayer Perceptron (MLP) classifier. We used evolutionary algorithms to find the best design that could attain the highest accuracy in order to enhance its performance. Backpropagation was used in the training of the MLP classifier. A homogeneous distribution of random weights between 0 and 1 was used to initialize the network.

A fivefold cross-validation method was used to assess how well the MLP will perform on new data. This was important because there weren't enough cases in the dataset, and training with 90% of them might have resulted in overfitting. As a result, utilizing a threefold cross-validation strategy helped to reduce the risk of overfitting while allowing for a more accurate evaluation of the model's generalization skills. The default MLP classifier parameters are shown in Table 2 along with the hyper-parameters that will be optimized by GA, the range of acceptable values, and gene locations.

The suggested parameter settings for the RCGAs when used with various datasets are presented in Table 3. To get the best results for each dataset, particular parameter settings might be necessary.

5.3 Results and Discussion

The performance of the genetic algorithm used to analyze the wine dataset across the first six generations is shown in Table 4. With a population size of 30, a simulated binary crossover probability of 0.9, and a polynomial mutation of 0.3, the algorithm performed fivefold cross-validation. The population's maximum and average fitness levels were seen to rise during the course of these generations. The minimum fitness value also demonstrated overall improvement. The fitness values' standard deviation dropped, pointing to a closer grouping around

Table 3 Suggested parameter settings for the RCGAs

Pc	Pm	Ps	MG	η
0.7	0.3	20	6	10
0.9	0.5	20	12	10
0.8	0.4	20	15	10
0.6	0.2	20	18	10

Pc: is cross-over probability, Pm: is mutation probability, Ps: is the population size, MG: is the maximum number of generations, η: index parameter of mutation

Table 4 Performance of the genetic algorithm applied to the wine datase

Generation	Max	Mean	Min	Std
0	0.949365	0.481011	0.331429	0.154401
1	0.949365	0.558561	**0.269841**	0.19579
2	0.949365	0.67091	0.331429	0.219346
3	**0.96619**	0.81737	0.315079	0.176233
4	0.96619	0.875741	0.315079	0.122659
5	0.96619	0.88163	0.315079	0.13712
6	0.96619	**0.933667**	0.836508	**0.026062**

the mean. These results point to the efficiency of the genetic algorithm in progressively improving population fitness and avoiding subpar solutions.

We may now look at Table5 to see the results of the genetic algorithm used on the hypothyroid dataset. The algorithm performed fivefold cross-validation using the same parameters as the wine dataset: a population size of 30, a simulated binary crossover probability of 0.8, and a polynomial mutation of 0.4. The maximum and average fitness values showed a substantial improvement across the generations, proving that the algorithm was successful in coming up with improved answers for this dataset. The fitness values' standard deviation dropped, indicating a decline in fitness variety among the population. These findings show how effectively the genetic algorithm improves population fitness and converges on the best answers for the hypothyroid dataset.

The results of the genetic algorithm used on the iris dataset are shown in Table 6.With a population size of 30, a simulated binary crossover probability of 0.9, and a polynomial mutation of 0.5, the algorithm conducted three cross-validations. Through the generations, the population's maximum and average fitness levels have increased. This shows how the genetic algorithm improved population fitness and converged to the best answers for the iris dataset.

Similar to this, Table 7 illustrates how the genetic algorithm used to analyze the cancer dataset improved after the first 16 generations. With a population size of 30, a simulated binary crossover probability of 0.9, and a polynomial mutation of 0.5, the algorithm conducted three cross-validations. The population's maximum and average fitness values grew, indicating that the genetic algorithm was successful in enhancing population fitness and convergent toward the best answers for the cancer dataset. Overall, our findings demonstrate the value of the genetic algorithm as an optimization tool for improving population fitness and achieving optimal results across a range of datasets.

Overall, the four datasets' neural network model outcomes were inconsistent. For the Wine dataset, a neural network model with six hidden layers, the relu activation function, the

Table 5 Performance of the genetic algorithm applied to the hypothyroid dataset

Generation	Max	Mean	Min	Std
0	0.98514	0.975034	0.952256	0.0123317
1	0.98514	0.949384	**0.0233947**	0.172049
2	0.98514	0.982412	0.972492	0.00236255
3	0.985458	0.983234	0.979451	0.00136439
4	0.985458	0.983602	0.981347	0.00117347
5	0.985458	0.982643	0.952256	0.0057651
6	0.985458	0.984087	0.981031	0.00122796
7	0.98609	0.983392	0.952256	0.00588725
8	0.98609	0.983993	0.979767	0.00150641
9	0.986405	0.98473	0.98198	0.00120309
10	**0.986721**	0.985226	0.982612	0.00104534
11	0.986721	0.985942	0.983877	**0.000724213**
12	0.986722	**0.986089**	0.982612	0.000848196

Table 6 Performance of the genetic algorithm applied to the iris dataset

Generation	Max	Mean	Min	Std
0	0.973333	0.701778	**0.286667**	0.248688
1	0.973333	0.840444	0.286667	0.189338
2	0.973333	0.913778	0.62	0.104505
3	0.973333	0.937556	0.666667	0.0879812
4	**0.98**	0.913556	0.286667	0.174216
5	0.98	0.927778	0.286667	0.140792
6	0.98	**0.971333**	0.953333	**0.00624203**

Table 7 Performance of the genetic algorithm applied to the cancer dataset

Generation	Max	Mean	Min	Std
0	0.936687	0.737209	0.372644	0.194603
1	0.936687	0.797649	0.372644	0.183507
2	0.943704	0.852265	0.530741	0.126529
3	0.943704	0.90183	0.572644	0.0820399
4	0.956047	0.916775	0.695777	0.0464411
5	0.956047	0.922997	0.850722	0.0273979
6	0.956047	0.930776	0.871728	0.0200354
7	0.957786	0.9265	0.808539	0.0390964
8	**0.964819**	0.933627	0.755426	0.0428287
9	0.964819	0.942896	0.746654	0.037727
10	0.964819	0.936345	0.789334	0.0429743
11	0.964819	0.939916	0.755426	0.039927
16	0.964819	**0.964233**	**0.952507**	**0.00237335**

Table 8 Optimal configuration of hyperparameters applied to each dataset

Dataset	Architecture	A.function	Solver	L.rate	R.parameter	Accuracy
Wine	(6,)	relu	lbfgs	0.30	Constant	0.96
Hypothyroide	(9,)	relu	lbfgs	1.87	Constant	0.9867
Iris	(11,)	tanh	lbfgs	0.48	Invscaling	0.98
Breast cancer	(10,)	relu	lbfgs	1.58	Invscaling	0.96

lbfgs solver, an alpha value of 0.30, and an adaptive learning rate produced the best results. This model had a 96.62% accuracy rate. Overall, it can be said that neural network models are capable of performing classification tasks; however, the best hyperparameters may change based on the particular dataset.

The neural network model with the best result for the Hypothyroid dataset featured 9 hidden layers, employed the relu activation function, the lbfgs solver, had an alpha value of 1.87, and had a constant learning rate. This model had a 98.67% accuracy rate.

For the Iris dataset, the best solution was achieved with a neural network model that had 13 hidden layers with 3 output neurons, using the 'tanh' activation function, 'lbfgs' solver, an alpha value of 0.48, and an invscaling learning rate. This model achieved an accuracy of 98%.

Finaly, for the Breast Cancer dataset, the best solution was achieved with a neural network model that had 10 hidden layers, used the 'relu' activation function, 'lbfgs' solver, an alpha value of 1.59, and an invscaling learning rate. This model achieved an accuracy of 96.48%.

The most accurate hyperparameters identified for each dataset are listed in Table 8, along with the MLP classifier's related accuracies.

We contrast the performance of our suggested model with that of numerous earlier research that made use of various categorization algorithms and methods. Table 9 presents the findings of this research. For all four datasets-wine, thyroid, iris, and breast cancer-our suggested approach outperformed earlier studies in terms of accuracy rates. These outcomes show how well our suggested model performs in precisely classifying various sorts of data.

In summary, the suggested strategy performs better than other methods on a variety of datasets, starting with the Wine dataset:

The accuracy of the Wine BP model, which uses a backpropagation neural network with one hidden layer made up of 10 neurons, was 95.0%. This shows that the model was successful in learning from and accurately classifying the wine data. With the identical design of 1 hidden layer and 10 neurons, the GA Weights model, which uses a genetic algorithm, achieved an accuracy of 96.00%, which is just a little bit higher. This implies that using a genetic algorithm optimization strategy helped the model perform better on the Wine dataset. With the same 1 hidden layer and 10 neurons architecture, the GABP model, which combines the genetic algorithm with backpropagation approaches, also managed to reach an accuracy of 96.00%. This suggests that when compared to using backpropagation alone, the combined strategy produced better outcomes. The MLP-GA model, which included 2 hidden layers with 13 neurons and a genetic algorithm for weight optimization, had a lower accuracy of 77.45%. This outcome shows that the optimization strategy and architecture used might not have been appropriate for this particular dataset. The accuracy of the P.method model, which had hyper-parameters of 1 hidden layer and 6 neurons, was 96.00%, which is comparable to that of the GA Weights and GABP models. This shows that the selected configuration of the hyper-parameters was successful in achieving high accuracy for the Wine dataset.

Table 9 Accuracy comparison of MLPGA+5 with several MLP based approach

Dataset	Methods	Optimization task	Architecture/neurons	Accuracy
Wine	BP [70]	Weights	1 H-L /10 N-L	95.00%
	GA [70]	Weights	1 H-L /10 N-L	96.00%
	GABP [70]	Weights	1 H-L /10 N-L	96.00%
	MLP-GA [71]	Weights and H-l	2 H-L /13 N-L	77.45%
	P.method	HPO	1 H-L /6 N-L	96.00%
Hypothyroid	MMLP [72]	FS	1 H-L /10 N-L	98.80%
	MLNN-LM(3×FC) [73]	Weights	50 neurons	92.96%
	MLNN-LM(10×FC) [73]	Weights	50 neurons	93.19%
	MLP-GA [74]	Weights	4–20 neurons	95%
	MLP-PSO [74]	Weights	4–20 neurons	85.5%
	P.method	HPO	1 H-L /9 N-L	98.67%
Iris	GE-BP [75]	–	–	96.6 ± 6.14%
	MLPGA+4 [76]	HPO	13.10 ± 11.30^{d}	98.87% ± 0.33%
	MLP-GA [2]	Architecture	3 H-L /4 N-L	97.30%
	GABP [77]	Weights	1 H-L /10 N-L	95.00%
	EBP [78]	Weights	–	97.30%
	P.method	Hyper-parameters	1 H-L /11 N-L	98.00%
Breast cancer	ANN [79]	Feature reducing	15-neuron ANN	99.4%
	SBS-BPPSO (10×CV) [80]	FS and weights	–	97.51%
	SBS-BPLM (10×CV) [80]	FS and weights	–	98.83%
	GONN[1] [81]	Weights and architecture	–	$98.24\%^{a}$ –$97.73\%^{b}$
	GONN[2] [81]	Weights and architecture	–	$99.63\%^{a}$ –$99.11\%^{b}$
	MLP [82]	HPO and FS	1 H-L /8 N-L	97.70%
	P.method	HPO	1 H-L /10 N-L	97.00%

H-L: Hidden layers, N-L: Neuron each layer, HPO: Hyper-parameters optimization, d. The MLP's average number of neurons, a: Best result obtained from one GP run, b: The average result obtained from 50 GP runs

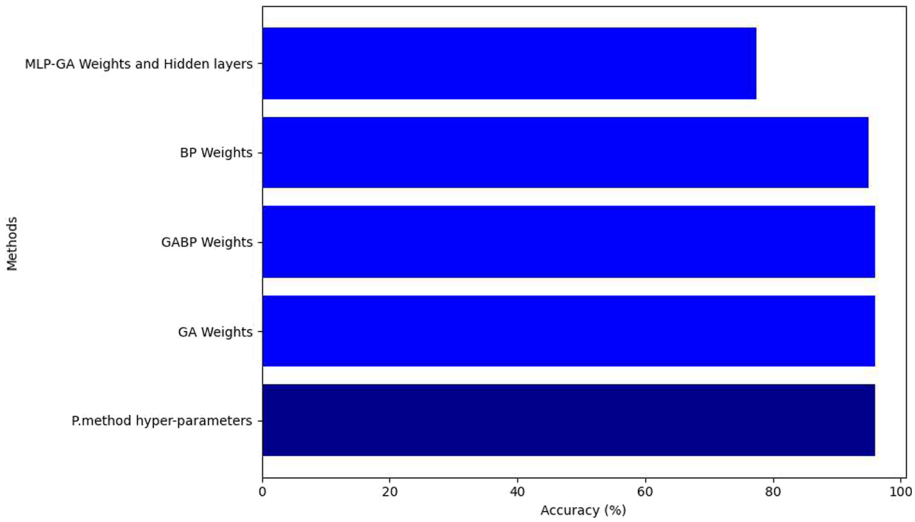

Fig. 10 Accuracy of different methods applied to wine dataset

The outstanding accuracy of 98.80% was attained by the Hypothyroid MMLP model, which makes use of a multi-layer perceptron with a feature scaling configuration of 1 high layer and 10 normal layers. This shows that the model effectively ingested and accurately identified the hypothyroid data. Comparatively, the accuracy of the MLNN-LM models with 50 neurons and 3 and 10 completely connected layers was 92.96% and 93.19%, respectively. These models show that adding more layers did not significantly increase accuracy, indicating that the added complexity did not significantly improve the dataset in question. A strong accuracy of 95% was attained by the MLP-GA model, which used genetic algorithm optimization with a range of 4 to 20 neurons. This shows that the weights of the model were successfully tuned by the genetic algorithm approach, leading to enhanced classification performance. The accuracy was 85.5% for the MLP-PSO model, which used particle swarm optimization with the same range of 4–20 neurons. This suggests that the genetic algorithm may have performed better than particle swarm optimization for this particular task. The P.method model had an accuracy of 98.67% with hyper-parameters set at 1 high layer and 9 normal layers. This outcome illustrates that the model's performance was successfully maximized by the selected hyper-parameter configuration, yielding high accuracy for the Hypothyroid dataset.

The Iris GE-BP model had a 96.6% accuracy and a 6.14% standard deviation for the Iris dataset. This shows that generally speaking, the model was successful in categorizing the many species of iris flowers. The relatively high standard deviation, however, points to some performance variation between various runs. The accuracy of the MLPGA+4 Hyperparameters model, on the other hand, was remarkable, coming in at 98.87% with a standard deviation of 11.30%. This model consistently produced good accuracy on the Iris dataset by combining Multi-Layer Perceptron with Genetic Algorithm optimization and four chosen hyperparameters. With three hidden layers and four normal layers, the MLP-GA Architecture model had an accuracy of 97.30%. This shows that a highly accurate classifier for the Iris dataset was produced using the selected architecture in combination with weight-tweaking genetic algorithm optimization.

The reported results for various methods in breast cancer classification are as follows: The ANN method's outstanding accuracy of 99.4% was attained using a 15-neuron ANN. A

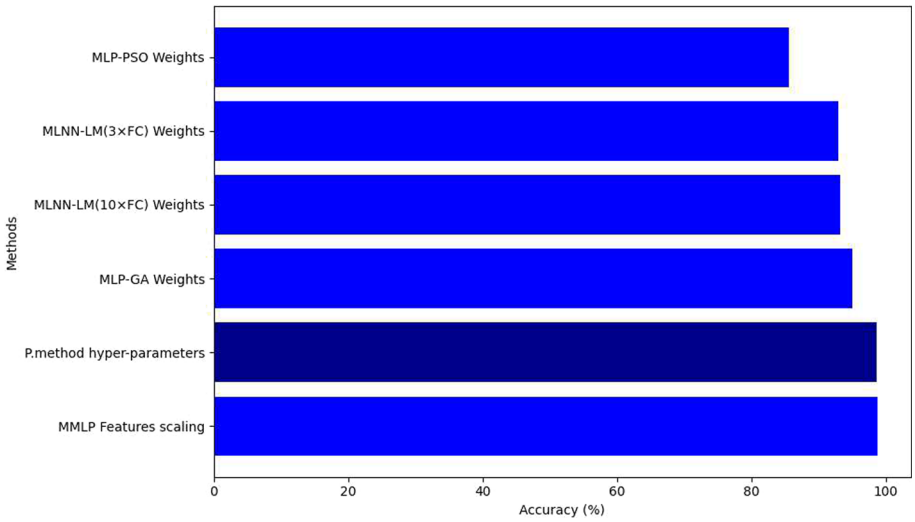

Fig. 11 Accuracy of different methods applied to hypothyroid dataset

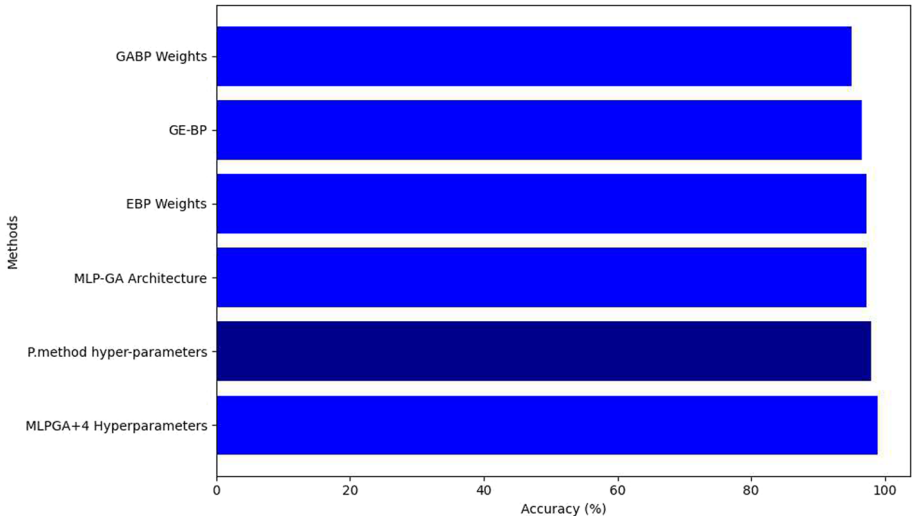

Fig. 12 Accuracy of different methods applied to iris dataset

97.51% accuracy was achieved using SBS-BPPSO (10 CV). SBS-BPLM (10 CV) attained a 98.83% accuracy. The accuracy of GONN1 was 98.24%. The accuracy of GONN2 was 99.63%. The accuracy of MLP hyper-parameters utilizing 1 hidden layer and 8 neurons was 97.70%. Last but not least, an accuracy of 97.00% was attained utilizing P.method hyper-parameters with 1 hidden layer and 10 neurons.

The results show that MLPRGA+5 performs differently on various datasets compared to the other approaches the authors have suggested, Figs. 10, 11, 12, and 13 depicts a summary of the results that were attained. Although it shows promise in some datasets, it might not work well in others. Further research and experimentation are advised to improve the results

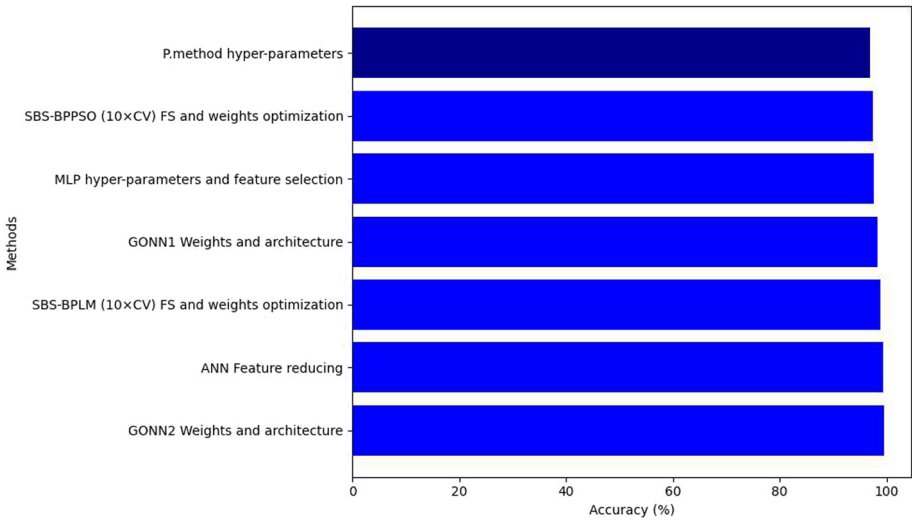

Fig. 13 Accuracy of different methods applied breast cancer dataset

in this particular dataset. This will aid in the method's improvement and optimization, maybe producing better results.

6 Conclusion

this study introduces a novel and highly effective approach MLPRGA+5 to optimize hyperparameters for Multilayer Perceptron (MLP) neural networks leveraging the Real-coded Genetic Algorithm (RCGA). Our work focuses on achieving an optimal balance between MLP accuracy and complexity. RCGA, along with specific operators like Simulated Binary Crossover (SBX) and Polynomial Mutation, offers a robust framework for navigating the intricate landscape of hyperparameters. The selection of RCGA is rooted in its ability to directly manage real-valued hyperparameters, aligning seamlessly with the continuous nature of these parameters. The synergy with SBX and Polynomial Mutation further empowers the algorithm to efficiently explore complex optimization spaces. This research extends the frontiers of hyperparameter tuning in MLP networks, emphasizing the broader utility of RCGAs in tackling intricate optimization challenges. By concurrently optimizing neural network architecture and hyperparameters, we provide a more efficient approach to classification tasks. Ultimately, our study underscores the potential of RCGAs as a valuable tool for machine learning and artificial intelligence, promoting advancements in deep learning and effective solutions to real-world problems.

References

1. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
2. Ramchoun H, Ghanou Y, Ettaouil M, Janati Idrissi MA (2016) Multilayer perceptron: architecture optimization and training
3. Yang L, Shami A (2020) On hyperparameter optimization of machine learning algorithms: theory and practice. Neurocomputing 415:295–316
4. Elshawi R, Maher M, Sakr S (2019) Automated machine learning: state-of-the-art and open challenges. arXiv preprint arXiv:1906.02287
5. Diaz GI, Fokoue-Nkoutche A, Nannicini G, Samulowitz H (2017) An effective algorithm for hyperparameter optimization of neural networks. IBM J Res Dev 61(4/5):9–1
6. Yu T, Zhu H (2020) Hyper-parameter optimization: a review of algorithms and applications. arXiv preprint arXiv:2003.05689
7. Nematzadeh S, Kiani F, Torkamanian-Afshar M, Aydin N (2022) Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: a bioinformatics study on biomedical and biological cases. Comput Biol Chem 97:107619
8. Weerts HJ, Mueller AC, Vanschoren J (2020) Importance of tuning hyperparameters of machine learning algorithms. arXiv preprint arXiv:2007.07588 (2020)
9. Sun D, Wen H, Wang D, Xu J (2020) A random forest model of landslide susceptibility mapping based on hyperparameter optimization using Bayes algorithm. Geomorphology 362:107201
10. Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. Adv Neural Inf Process Syst 24
11. Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(2)
12. Eggensperger K, Feurer M, Hutter F, Bergstra J, Snoek J, Hoos H, Leyton-Brown K, et al (2013) Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NIPS Workshop on Bayesian optimization in theory and practice, vol 10
13. Lessmann S, Stahlbock R, Crone SF (2005) Optimizing hyperparameters of support vector machines by genetic algorithms. In: IC-AI, vol 74, p 82
14. Claesen M, Simm J, Popovic D, Moreau Y, De Moor B (2014) Easy hyperparameter search using optunity. arXiv preprint arXiv:1412.1114
15. Eggensperger K, Hutter F, Hoos H, Leyton-Brown K (2015) Efficient benchmarking of hyperparameter optimizers via surrogates. In: Proceedings of the AAAI conference on artificial intelligence, vol 29
16. Chen C-T, Wu C-K, Hwang C (2008) Optimal design and control of cpu heat sink processes. IEEE Trans Compon Packag Technol 31(1):184–195
17. Chen C-T, Chuang Y-C (2010) An intelligent run-to-run control strategy for chemical-mechanical polishing processes. IEEE Trans Semicond Manuf 23(1):109–120
18. Dyer JD, Hartfield RJ, Dozier GV, Burkhalter JE (2012) Aerospace design optimization using a steady state real-coded genetic algorithm. Appl Math Comput 218(9):4710–4730
19. Tsai C-W, Lin C-L, Huang C-H (2010) Microbrushless dc motor control design based on real-coded structural genetic algorithm. IEEE/ASME Trans Mechatron 16(1):151–159
20. Valarmathi K, Devaraj D, Radhakrishnan T (2009) Real-coded genetic algorithm for system identification and controller tuning. Appl Math Model 33(8):3392–3401
21. Chuang Y-C, Chen C-T, Hwang C (2015) A real-coded genetic algorithm with a direction-based crossover operator. Inf Sci 305:320–348
22. Eshelman LJ, Schaffer JD (1993) Real-coded genetic algorithms and interval-schemata. Found Genet Algorithms 2:187–202 (**Elsevier,????**)
23. Tsutsui S, Goldberg DE (2001) Search space boundary extension method in real-coded genetic algorithms. Inf Sci 133(3–4):229–247
24. Beyer H-G, Deb K (2001) On self-adaptive features in real-parameter evolutionary algorithms. IEEE Trans Evol Comput 5(3):250–270
25. Ripon KSN, Kwong S, Man K-F (2007) A real-coding jumping gene genetic algorithm (rjgga) for multiobjective optimization. Inf Sci 177(2):632–654

26. Chen Z-Q, Yin Y-F (2012) An new crossover operator for real-coded genetic algorithm with selective breeding based on difference between individuals. In: 2012 8th international conference on natural computation. IEEE, pp 644–648
27. Yoon Y, Kim Y-H, Moraglio A, Moon B-R (2012) A theoretical and empirical study on unbiased boundary-extended crossover for real-valued representation. Inf Sci 183(1):48–65
28. Deb K, Anand A, Joshi D (2002) A computationally efficient evolutionary algorithm for real-parameter optimization. Evol Comput 10(4):371–395
29. Deb K, Beyer H-G (2001) Self-adaptive genetic algorithms with simulated binary crossover. Evol Comput 9(2):197–221
30. Deb K, Agrawal RB et al (1995) Simulated binary crossover for continuous search space. Complex Syst 9(2):115–148
31. Deb K, Sindhya K, Okabe T (2007) Self-adaptive simulated binary crossover for real-parameter optimization. In: Proceedings of the 9th annual conference on genetic and evolutionary computation, pp 1187–1194
32. Dobnikar A, Steele NC, Pearson DW, Albrecht RF, Deb K, Agrawal S (1999) A niched-penalty approach for constraint handling in genetic algorithms. In: Artificial neural nets and genetic algorithms: proceedings of the international conference in Portorož, Slovenia. Springer, pp 235–243
33. Zeng G-Q, Chen J, Li L-M, Chen M-R, Wu L, Dai Y-X, Zheng C-W (2016) An improved multi-objective population-based extremal optimization algorithm with polynomial mutation. Inf Sci 330:49–73
34. Asuncion A, Newman D (2007) UCI machine learning repository. Irvine, CA, USA
35. Yeung DS, Li J-C, Ng WW, Chan PP (2015) Mlpnn training via a multiobjective optimization of training error and stochastic sensitivity. IEEE Trans Neural Netw Learn Syst 27(5):978–992
36. Chiroma H, Abdulkareem S, Abubakar A, Herawan T (2017) Neural networks optimization through genetic algorithm searches: a review. Appl Math Inf Sci 11(6):1543–1564
37. Wang Z-H, Gong D-Y, Li X, Li G-T, Zhang D-H (2017) Prediction of bending force in the hot strip rolling process using artificial neural network and genetic algorithm (ann-ga). Int J Adv Manuf Technol 93:3325–3338
38. Gill EJ, Singh EB, Singh ES (2010) Training back propagation neural networks with genetic algorithm for weather forecasting. In: IEEE 8th international symposium on intelligent systems and informatics. IEEE, pp 465–469
39. Al-qaness MA, Ewees AA, Elaziz MA, Samak AH (2022) Wind power forecasting using optimized dendritic neural model based on seagull optimization algorithm and aquila optimizer. Energies 15(24):9261
40. Ansari S, Alnajjar KA, Saad M, Abdallah S, El-Moursy AA (2022) Automatic digital modulation recognition based on genetic-algorithm-optimized machine learning models. IEEE Access 10:50265–50277
41. Domashova J, Yakimov D, Bredikhin D, Gorbunov K, Slavik R, Kadyrov I (2022) Detection and analysis of atypical stock transactions with possible misuse of insider information and market manipulation: methods and models. Proc Comput Sci 213:165–174
42. Sreedharan M, Khedr AM, El Bannany M (2020) A multi-layer perceptron approach to financial distress prediction with genetic algorithm. Autom Control Comput Sci 54:475–482
43. Daviran M, Shamekhi M, Ghezelbash R, Maghsoudi A (2023) Landslide susceptibility prediction using artificial neural networks, svms and random forest: hyperparameters tuning by genetic optimization algorithm. Int J Environ Sci Technol 20(1):259–276
44. Kumar P, Batra S, Raman B (2021) Deep neural network hyper-parameter tuning through twofold genetic approach. Soft Comput 25:8747–8771
45. Mohan B, Badra J (2023) A novel automated superlearner using a genetic algorithm-based hyperparameter optimization. Adv Eng Softw 175:103358
46. Abdollahi J, Nouri-Moghaddam B (2022) Hybrid stacked ensemble combined with genetic algorithms for diabetes prediction. Iran J Comput Sci 5(3):205–220
47. Wang H, Moayedi H, Kok Foong L (2021) Genetic algorithm hybridized with multilayer perceptron to have an economical slope stability design. Eng Comput 37:3067–3078
48. Tayebi M, El Kafhali S (2022) Performance analysis of metaheuristics based hyperparameters optimization for fraud transactions detection. Evolut Intell, 1–19
49. Arukonda S, Cheruku R (2023) A novel diversity-based ensemble approach with genetic algorithm for effective disease diagnosis. Soft Comput, 1–20
50. Martínez-Comesaña M, Ogando-Martínez A, Troncoso-Pastoriza F, López-Gómez J, Febrero-Garrido L, Granada-Álvarez E (2021) Use of optimised mlp neural networks for spatiotemporal estimation of indoor environmental conditions of existing buildings. Build Environ 205:108243
51. Ettaouil M, Ghanou Y (2009) Neural architectures optimization and genetic algorithms. Wseas Trans Comput 8(3):526–537

52. Sezer OB, Gudelek MU, Ozbayoglu AM (2020) Financial time series forecasting with deep learning: a systematic literature review: 2005–2019. Appl Soft Comput 90:106181
53. Ogundokun RO, Misra S, Douglas M, Damaševičius R, Maskeliūnas R (2022) Medical internet-of-things based breast cancer diagnosis using hyperparameter-optimized neural networks. Fut Internet 14(5):153
54. Ji M, Zhang K, Wu Q, Deng Z (2020) Multi-label learning for crop leaf diseases recognition and severity estimation based on convolutional neural networks. Soft Comput 24:15327–15340
55. Kumar N, Susan S (2021) Particle swarm optimization of partitions and fuzzy order for fuzzy time series forecasting of covid-19. Appl Soft Comput 110:107611
56. Jamil M, Yang X-S (2013) A literature survey of benchmark functions for global optimisation problems. Int J Math Modell Numer Optim 4(2):150–194
57. Vijaya J, Sivasankar E (2019) An efficient system for customer churn prediction through particle swarm optimization based feature selection model with simulated annealing. Clust Comput 22:10757–10768
58. Sun S, Cao Z, Zhu H, Zhao J (2019) A survey of optimization methods from a machine learning perspective. IEEE Trans Cybern 50(8):3668–3681
59. McCarl BA, Spreen TH (1997) Applied mathematical programming using algebraic systems. Cambridge, MA
60. Lorenzo PR, Nalepa J, Kawulok M, Ramos LS, Pastor JR (2017) Particle swarm optimization for hyper-parameter selection in deep neural networks. In: Proceedings of the genetic and evolutionary computation conference, pp 481–488
61. Gambella C, Ghaddar B, Naoum-Sawaya J (2021) Optimization problems for machine learning: a survey. Eur J Oper Res 290(3):807–828
62. Lucasius CB, Kateman G (1989) Application of genetic algorithms in chemometrics. In: Proceedings of the 3rd international conference on genetic algorithms, pp 170–176
63. Davis L (1989) Adapting operator probabilities in genetic algorithms. In: Proceedings of the 3rd international conference on genetic algorithms
64. Davis L (1991) Handbook of genetic algorithms
65. Wright AH (1991) Genetic algorithms for real parameter optimization. Found Genet Algorithms 1:205–218 (**Elsevier,????**)
66. Vignaux GA, Michalewicz Z (1991) A genetic algorithm for the linear transportation problem. IEEE Trans Syst Man Cybern 21(2):445–452
67. Mühlenbein H, Schlierkamp-Voosen D (1993) Predictive models for the breeder genetic algorithm i. Continuous parameter optimization. Evol Comput 1(1):25–49
68. Chuang Y-C, Chen C-T, Hwang C (2016) A simple and efficient real-coded genetic algorithm for constrained optimization. Appl Soft Comput 38:87–105
69. Lichman M et al (2013) UCI machine learning repository. Irvine, CA, USA
70. Ding S, Su C, Yu J (2011) An optimizing bp neural network algorithm based on genetic algorithm. Artif Intell Rev 36:153–162
71. Ramchoun H, Idrissi MJ, Ghanou Y, Ettaouil M (2017) New modeling of multilayer perceptron architecture optimization with regularization: an application to pattern classification. IAENG Int J Comput Sci 44(3):261–269
72. Hosseinzadeh M, Ahmed OH, Ghafour MY, Safara F, Hama HK, Ali S, Vo B, Chiang H-S (2021) A multiple multilayer perceptron neural network with an adaptive learning algorithm for thyroid disease diagnosis in the internet of medical things. J Supercomput 77:3616–3637
73. Temurtas F (2009) A comparative study on thyroid disease diagnosis using neural networks. Expert Syst Appl 36(1):944–949
74. Sharifi A, Alizadeh K (2021) Comparison of the particle swarm optimization with the genetic algorithms as a training for multilayer perceptron technique to diagnose thyroid functional disease. Shiraz E-Med J 22(1)
75. Soltanian K, Tab FA, Zar FA, Tsoulos I (2013) Artificial neural networks generation using grammatical evolution. In: 2013 21st Iranian conference on electrical engineering (ICEE). IEEE, pp 1–5
76. Itano F, Sousa MAdA, Del-Moral-Hernandez E (2018) Extending mlp ann hyper-parameters optimization by using genetic algorithm. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 . IEEE
77. Papakostas G, Boutalis Y, Samartzidis S, Karras D, Mertzios B (2004) Combining backpropagation and genetic algorithms to train neural networks. In: IWSSIP 2005 (Proceedings of 12th international workshop on systems, signals & image processing), pp 169–175
78. Patil BG, Subbaraman S (2012) Svd-ebp algorithm for iris pattern recognition. arXiv preprint arXiv:1204.2062
79. Chaurasia V, Pal S (2020) Applications of machine learning techniques to predict diagnostic breast cancer. SN Comput Sci 1(5):270

80. Huang M-L, Hung Y-H, Chen W-Y (2010) Neural network classifier with entropy based feature selection on breast cancer diagnosis. J Med Syst 34:865–873
81. Bhardwaj A, Tiwari A (2015) Breast cancer diagnosis using genetically optimized neural network model. Expert Syst Appl 42(10):4611–4620
82. Al-Shargabi B, Alshami F, Alkhawaldeh R (2019) Enhancing multi-layer perception for breast cancer prediction. Int J Adv Sci Technol