Journal of Mathematical Modeling
Vol. 12, No. 4, 2024, pp. 603–622. Research Article

JMM

Multi–objective model for architecture optimization and training of radial basis function neural networks

Taoufyq Elansari*, **Mohammed Ouanan, Hamid Bourray**

TSI Team, Department of Computer Sciences, Faculty of Sciences, Moulay Ismail University, B.P. 11201 Zitoune, Meknes, Morocco
Email(s): t.elansari@edu.umi.ac.ma, m.ouanan@umi.ac.ma, hbourrayh@yahoo.fr

Abstract. Radial Basis Function Neural Network (RBFNN) is a type of artificial neural networks used for supervised learning. They rely on radial basis functions (RBFs), nonlinear mathematical functions employed to approximate complex nonlinear data. Determining the architecture of the network is challenging, impacting the achievement of optimal learning and generalization capacities. This paper presents a multi–objective model for optimizing and training RBFNN architecture. The model aims to fulfill three objectives: the first is the summation of distances between the input vector and the corresponding center for the neurons in the hidden layer. The second objective is the global error of the RBFNN, defined as the discrepancy between the calculated output and the desired output. The third objective is the complexity of the RBFNN, quantified by the number of neurons in the hidden layer. This innovative approach utilizes multiple objective simulated annealing to identify optimal parameters and hyperparameters for neural networks. The numerical results provide accuracy and reliability of the theoretical results discussed in this paper, as well as advantages of the proposed approach.

Keywords: Radial basis function neural network, multiple objective simulated annealing, universal approximation.
AMS Subject Classification 2010: 90C29, 68T20.

1 Introduction

Multi–objective Optimization (MO) is a powerful tool for addressing problems that encompass multiple, often conflicting objectives [1,7]. Its applications span various fields, including engineering, manufacturing, finance, and economics. The primary goal of MO is to attain an optimal solution that simultaneously satisfies all defined objectives [43].

In recent years, data processing has emerged as a significant challenge across various domains in engineering sciences. Tasks such as classification [10,27], identification [40,41], and approximation [24]

*Corresponding author
Received: 30 January 2024 / Revised: 6 May 2024 / Accepted: 6 June 2024
DOI: 10.22124/jmm.2024.26636.2345

are ubiquitous challenges in various domains. For example, in the field of sensor development, methodologies such as rational design of field–effect sensors using partial differential equations, Bayesian inversion, and artificial neural networks [19] play a central role. Similarly, advances such as hyperparameter optimization of stacked asymmetric autoencoders for automatic personality trait perception [44], strategies such as using layered training for semantic segmentation in self–driving cars [34], and techniques such as a multi–level adaptive Monte Carlo algorithm for the stochastic drift–diffusion–Poisson system [18] demonstrate the breadth of applications requiring sophisticated solutions. In each of these tasks, a dataset is employed to formulate nonparametric or parametric functions, which subsequently serve as generalizations for other data.

The Radial Basis Function Neural Network (RBFNN) is an artificial feedforward neural network introduced by Broomhead and Lowe in [5]. Subsequently, theorists have studied fundamental features of the RBF approximation, like density, uniqueness of interpolation, and rate of convergence [26, 29, 32]. An RBFNN has been proved to be able to approximate any continuous multivariate function on a compact domain in an empirically arbitrary manner if provided with a reasonable number of radial function neurons [26, 32]. In recent years, RBFNN has been used in many applications in different fields such as Differential equations [33], Nonlinear system identification [11], Biometric iris recognition [8], Spacecraft relative navigation [31], Fuel flow prediction [2], Predicting trihalomethane levels in drinking water [12], Vehicles using hypermutated firefly [13], Prediction of horizons wind speed [28]. The RBFNN is grounded in the concept of RBFs, non-linear mathematical functions employed for approximating complex non–linear data. The RBF networks comprise an input layer, a hidden layer, and an output layer. The hidden layer is constituted by a series of neurons, each featuring its own RBF function. Several approaches for learning RBFNNs were developed. The majority of them can be splitted into two phases: (i) training the centers and radii in the hidden layer; (ii) adjusting the connection weights connecting the hidden layer with the output layer. In this type of learning, there is a separation between the determination of the hidden layer parameters and the output layer weights during RBFNN training [39]. Another learning method is to train all RBFNN parameters in a fully supervised way, using the backpropagation (BP) or gradient descent algorithm [16]. This process has the inconvenience of a long time to learn and a high computing cost. While the RBFNN excels in approximating continuous functions and accurately identifying nonlinear systems, it is not without its set of challenges. One notable issue is the selection of the number of neurons in the hidden layer. Arbitrarily choosing this quantity can lead to suboptimal solutions. Opting for a larger number of neurons may result in overlearning and incorrect interpretations [14], whereas selecting too few neurons may lead to an inaccurate approximation of the data pattern [38]. In these training strategies, the network architecture or the quantity of neurons in the hidden layer is predetermined. If you want to know whether a particular network structure is appropriate for your application, you can only do it by trial and error. The learning strategies that include structure choice techniques have been presented by researchers in the following works [22, 30]. Nature-inspired metaheuristics have been deployed in different manners for training RBFNNs. Some have been applied to find one network parameter such as centers [20], others have optimized all parameters [35], while still others have sought to optimize all parameters as well as the RBFNN structure. The algorithms applied to the training of RBFNNs are the following: Genetic Algorithms (GA) [4], Particle Swarm Optimization (PSO) [13], Ant Colony Optimization (ACO) [6], Differential Evolution (DE) [42]. Selecting the optimal number of hidden layer in RBFNN is a critical aspect of designing its architecture. This decision influences the network's ability to learn and generalize from the data. Indeed, this problem not only has an extensive effect on convergence, it also influences the accuracy of the results received [15]. This work

contributes to solving the architectural selection problem by using MO. In this paper, we present a novel multi-objective model for the optimization and training of RBFNNs. This model contributes to hyper-parameter optimization in machine learning and establishes a theoretical foundation for the selection of RBFNN architecture and its training process. The model seeks to determine the maximum architecture initially and subsequently eliminates unnecessary neurons in the hidden layer. Traditional methods of experimentally choosing the architecture often consume more time and may lead to overfitting or under-fitting issues. Our proposed model addresses this challenge by formulating a non-linear MO optimization problem with mixed variables. We employ an approach based on multiple objective simulated annealing to determine optimal weights and an effective architecture.

This paper is structured as follows: In Section 2, preliminaire and related work are summarized. In Section 3, a new modeling of multi-objective optimization of RBFNN training is presented. In Section 4, solving the obtained model using a new approach based on simulated annealing is presented. Experimental study and discussion of the results are presented in Section 5.

2 Preliminaire and related work

2.1 Radial basis function neural networks

The RBFNN, depicted in Fig 1, operates as a feedforward network [9]. Each neuron in the hidden layer adopts an RBF $\phi(x)$ as its activation function with $\phi_k(x) = \phi(\|\boldsymbol{x} - \boldsymbol{c}_k\|)$, where $\boldsymbol{c}_k$ denotes the center of the k^{th} neuron. The network's output is a linear combination of the hidden layer's output. Given an input $\boldsymbol{x}$, the RBFNN's output is defined by the equation:

$$\hat{y}_i(\boldsymbol{x}) = \sum_{j=1}^{N} w_{ji}\phi\left(\left\|\boldsymbol{x} - \boldsymbol{c}_j\right\|\right) \quad i = 1, \ldots, m, \tag{1}$$

where

- N: The total number of neurons present in the hidden layer.
- m: The total number of neurons in the output layer.
- $\hat{y}_i(x)$: is the output of neuron i in the RBFNN output layer.
- w_{ji}: is the connection weight of j^{th} neurons in the hidden layer to i^{th} neurons in the output layer.
- $\|\cdot\|$: Euclidean norm.

2.1.1 Learning RBFNN

The objective of the learning process is to discover a method for adapting the parameters of the network when presenting examples from the training set. The parameter matrix of an RBFNN is denoted as $\mathbf{P} = [C, \sigma, W]$, encompassing centers, radii, and linear parameters. These parameters are typically adjusted to minimize the cost function for a set of n pairs of patterns, denoted as $X = \left\{(x_p, y_p)_{1 \leq p \leq n}\right\}$:

$$J(\mathbf{P}) = \frac{1}{n}\left\|\mathbf{Y} - \hat{f}_{RBF}(\mathbf{X}, \mathbf{C}, \sigma, W)\right\|^2. \tag{2}$$

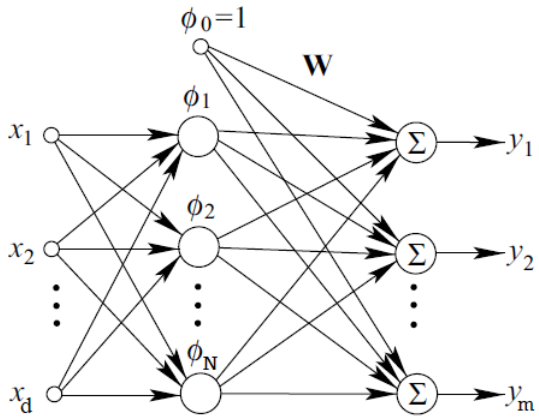

Figure 1: Architecture of RBF neural networks.

In the literature, two approaches to find RBFNN parameters have been proposed. The first strategy relies on supervised or direct methods that employ computationally expensive algorithms such as the gradient algorithm [16]. The second approach adopts a hybrid training algorithm for RBF networks [9].

2.1.2 Hybrid algorithms

Unsupervised party: The parameters of the hidden layer $C = (c_1, \ldots, c_N)$ and $\sigma = (\sigma_1, \ldots, \sigma_N)$ are obtained in this part of the algorithm. A competitive neural network is a single layer of N neurons. Their weights are $c_1, \ldots, c_N \in \mathbb{R}^d$. We divide the entry space into N partitions. $\mathscr{E}_1, \ldots, \mathscr{E}_N$, in which $\forall j = 1, \ldots, N$ we have

$$\mathscr{E}_j = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \left\| \mathbf{x} - \mathbf{c}_j \right\| = \min_{i=1,\ldots,N} \left\| \mathbf{x} - \mathbf{c}_i \right\| \right\}. \tag{3}$$

Each weight c_j represents a prototype vector which is representative of the $\mathscr{E}_j$ region. Several methods are proposed to adapt the centers and the scaling parameters, such as the K–means clustering [36] and the self–organizing feature map [21].

Supervised party: In this segment of the algorithm, we fine–tune the linear parameters, denoted as W, of the output layer. Given a set of n pattern pairs $X = \left\{ (x_p, y_p)_{1 \leq p \leq n} \right\}$, the weights matrix W is determined through the minimization of the error function

$$J(\mathbf{P}) = \frac{1}{n} \left\| \mathbf{Y} - \hat{f}_{RBF}(\mathbf{X}, \mathbf{C}, \sigma, W) \right\|_2^2. \tag{4}$$

The explicit solution is derived as follows

$$\hat{W} = \left[\Phi \Phi^t \right]^{-1} \Phi \mathbf{Y}. \tag{5}$$

The key components are

- $W = [w_1, \ldots, w_m]$ is an $N \times m$ weight matrix, where $w_i = (w_{1i}, \ldots, w_{Ni})^T$ is the weight vector for neuron i in the output layer.

- $\Phi = [\phi_1, \ldots, \phi_N]$ is an $N \times d$ matrix, where $\phi_p = (\phi_{p,1}, \ldots, \phi_{p,N})^T$ represents the output of the hidden layer for the p^{th} sample, and $\phi_{p,k} = \phi\left(\|x_p - c_k\|\right)$.
- $Y = [y_1, y_2, \ldots, y_d]$ is an $m \times d$ matrix, and $y_p = (y_{p,1}, \ldots, y_{p,m})^T$ is the desired output for the p^{th} sample in the training database.
- $\hat{f}_{RBF}(\mathbf{X}, \mathbf{C}, \sigma, W) = \Phi^t W$ represents the output of the RBFNN.

2.2 Multiple objective simulated annealing (MOSA)

2.2.1 Simulated annealing (SA)

This method is used to determine the global minimum of a given cost function that may have several minima initially. It was developed by IBM scientists and is inspired by the physical process used in metallurgy. In the Algorithm 1, we first randomly choose a starting point x, and calculate a neighborhood of this point y using the operator Neighbor(x). We evaluate this neighboring point and compute the difference with the original point, $\Delta F = F(y) - F(x)$; if the difference is negative, we take y as the new starting point. If the difference is positive, we may choose y as the new starting point, but only with probability $e^{-\frac{\Delta F}{T}}$. During the execution of this algorithm, the temperature T is reduced. We repeat these steps until our system is stabilized.

Algorithm 1 : Simulated Annealing

Require: T, x, $ItrMax$, ΔT

$x^* \leftarrow x$
while $T \geq \Delta T$ **do**
  for $k = 1$ **to** $ItrMax$ **do**
    $y \leftarrow$ Neighbor(x)
    $\Delta F \leftarrow F(y) - F(x)$
    if $\Delta F < 0$ **or** rand$(0,1) <$ exp$(-\frac{\Delta F}{T})$ **then**
      $x \leftarrow y$
    end if
    if $F(x) < F(x^*)$ **then**
      $x^* \leftarrow x$
    end if
  end for
  $T \leftarrow \alpha(T)$
end while
return x^*

We are now going to present the MOSA.

2.2.2 Multiple objective simulated annealing

This method is an extension of SA designed to address MO problems, aiming to identify the compromise surface. The technique is introduced in [37]. Initially, we define a set of functions that determine the

probability of accepting a suboptimal solution for each objective function:

$$\pi_k = \begin{cases} \exp\left(-\frac{\Delta f_k}{T_n}\right), & \text{if } \Delta f_k > 0, \\ 1, & \text{if } \Delta f_k \leq 0, \end{cases} \tag{6}$$

where T_n represents the temperature at iteration n, f_k denotes the k^{th} objective function, x_n is the solution obtained at iteration n, y stands for the actual solution at iteration n, and Δf_k calculates the difference between the k^{th} objective function values of y and x_n.

Subsequently, after calculating all these individual probabilities, we combine them to obtain the overall probability of acceptance using the formula $p = \prod_{k=1}^{N} \pi_k^{\lambda_k}$, where λ_k represents a weighting coefficient associated with the k^{th} objective function and N is the number of objective functions.
Finally, the selection of a new solution is made according to the following rule

- if $\Delta f_{eq} \leq 0$, then: $x_{n+1} = y$.
- if $\Delta f_{eq} > 0$, then: $x_{n+1} = \begin{cases} y, & \text{with probability } p, \\ x_n, & \text{with probability } 1-p, \end{cases}$

where $f_{eq}(x) = \sum_{i=1}^{N} \lambda_i f_i(x)$ and $\Delta f_{eq} = f_{eq}(x_{n+1}) - f_{eq}(x_n)$.

3 A new modeling of neural architecture optimization and training

3.1 Notations

- n: Number of observations in the database.
- d: Number of neurons present in the input layer.
- N: Number of neurons in the hidden layer.
- m: Number of neurons in the output layer.
- X: Input data of the neural network.
- ϕ: Activation function.
- c^i: The center of the i^{th} neuron in the hidden layer.
- σ_i: The radius of the i^{th} neuron in the hidden layer.
- W: Network weights for output layer.
- $\hat{Y}$: Computed output by the neural network.
- Y: Desired output, where $Y^k = \left(y_1^k, \ldots, y_m^k\right)$ for $k = 1, \ldots, n$.
- $X = \left\{x^1, \ldots, x^n\right\}$: Input data of training base where $x^k = \left(x_1^k, \ldots, x_d^k\right)$ for $k = 1, \ldots, n$.
- $u_{i,j}$: The binary variable for i $= 1, \ldots,$ n and j $= 1, \ldots,$ N, $u_{i,j} = 1$ if the i^{th} example is assigned to j^{th} neuron, and $u_{i,j} = 0$ else.
- v_j: The binary variable for $j = 1, \ldots, N, v_j$ takes 0 if the j^{th} neuron is deleted, otherwise $v_j = 1$ if j^{th} neuron is used.

3.1.1 Output of the neural network

The RBFNN output for the example x^k is calculated by this expression

$$\hat{Y}^k = \begin{pmatrix} \hat{Y}_1^k \\ \vdots \\ \hat{Y}_j^k \\ \vdots \\ \hat{Y}_m^k \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} v_i w_{i1} \phi \left(\left\| \frac{\sum_{h=1}^{n} u_{hi} x^h}{\sum_{h=1}^{n} u_{hi}} - x^k \right\|, \sigma_i \right) \\ \vdots \\ \sum_{i=1}^{N} v_i w_{ij} \phi \left(\left\| \frac{\sum_{h=1}^{n} u_{hi} x^h}{\sum_{h=1}^{n} u_{hi}} - x^k \right\|, \sigma_i \right) \\ \vdots \\ \sum_{i=1}^{N} v_i w_{im} \phi \left(\left\| \frac{\sum_{h=1}^{n} u_{hi} x^h}{\sum_{h=1}^{n} u_{hi}} - x^k \right\|, \sigma_i \right) \end{pmatrix}, \tag{7}$$

where $\phi(x, \sigma) = e^{\frac{-x^2}{2\sigma^2}}$.

3.2 Objectives functions

In our model, we need to consider three objective functions. The first one involves calculating the sum of the differences between the input vector and the corresponding centers of the utilized neurons

$$F_1(u, v) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{N} v_j u_{i,j} \left\| x^i - \frac{\sum_{h=1}^{n} u_{hj} x^h}{\sum_{h=1}^{n} u_{hj}} \right\|. \tag{8}$$

The second objective is to minimize the overall error of the RBFNN, which is defined as the difference between the output computed by the network and the intended output. This error is typically presented as the discrepancy between the predicted output and the actual output for a given set of input data

$$F_2(u, v, x, \sigma, W) = \frac{1}{n} \sum_{i=1}^{n} \left\| \hat{Y}^i(u, v, \sigma, W) - Y^i \right\|. \tag{9}$$

The third objective is the complexity of RBFNN, defined as the number of neurons in the hidden layer normalized by the maximum number of neurons in the hidden layer and multiplied by the database size. This is presented as follows

$$F_3(v) = \frac{1}{n \times m} \sum_{j=1}^{N} v_j. \tag{10}$$

3.3 Constraints

Assignment constraints The following constraints ensure that each example is assigned to one neuron:

$$\sum_{j=1}^{N} u_{i,j} = 1, \quad \text{for } i = 1, \ldots, n. \tag{11}$$

This is a fundamental requirement for most clustering tasks, ensuring that every data point is accounted for in the model.

Equilibrium constraint The following constraints ensure that if a neuron is not used, no example is assigned to it

$$(1-v_j)\sum_{1=1}^{n} u_{i,j} = 0, \quad \text{for } j = 1,\ldots,N. \tag{12}$$

This makes sense because if a neuron is not active, it should not influence the assignment of any examples. It effectively enforces a kind of "if–then" relationship between the activation of a neuron and the assignment of examples to it.

Use of neurons The below constraints ensure that if assignment for each neuron is used, then there is at least one example that is affected

$$\sum_{j=1}^{N} v_j \times \prod_{i=1}^{n} (1-u_{i,j}) = 0. \tag{13}$$

This is important for ensuring that activated neurons actually contribute to the model's ability in clustering data to calculate the center of RBF in this neuron. If a neuron is 'on,' it should have some examples associated with it; otherwise, it would be redundant or unnecessary.

Set of feasible solutions We denote by S the set of feasible solutions defined by $S = \{(v,u,\sigma,W) : g_{1i}(u) = 0\ \forall i \in \{1,\ldots,n\}, g_2(v,u) = 0,\ g_{3j}(u,v) = 0\ \forall j \in \{1,\ldots,N\}, v \in \{0,1\}^N,\ u \in \{0,1\}^N \times \{0,1\}^n, \sigma \in \mathbb{R}_*^N, W \in \mathbb{R}^m \times \mathbb{R}^N\}$, where $g_{1i}(u) = 1 - \sum_{j=1}^{N} u_{i,j}$, $g_2(u,v) = \sum_{j=1}^{N} v_j \times \prod_{i=1}^{n} (1-u_{i,j})$ and $g_{3j}(u,v) = (1-v_j)\sum_{1=1}^{n} u_{i,j}$.

3.4 Optimization model

The architecture optimization and training model of RBFNN can be formulated as follows:

$$(P)\begin{cases}
\min \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{N} v_j u_{i,j} \left\| x^i - \frac{\sum_{h=1}^{n} u_{hj} x^h}{\sum_{h=1}^{n} u_{hj}} \right\|, \\
\min \frac{1}{n}\sum_{i=1}^{n} \left\| \hat{Y}^i(u,v,\sigma,W) - Y^i \right\|, \\
\min \frac{1}{m\times n}\sum_{j=1}^{N} v_j, \\
\text{Subject to} \\
\quad \sum_{j=1}^{N} v_j \times \prod_{i=1}^{n} (1-u_{i,j}) = 0, \\
\quad \sum_{j=1}^{N} u_{i,j} = 1, & \forall i = 1,\ldots,n, \\
\quad (1-v_j)\sum_{i=1}^{n} u_{i,j} = 0, & \forall j = 1,\ldots,N, \\
\quad u_{i,j} \in \{0,1\}, & \forall i = 1,\ldots,n, \quad \forall j = 1,\ldots,N, \\
\quad v_j \in \{0,1\}, & \forall j = 1,\ldots,N, \\
\quad \sigma_j \in \mathbb{R}_+^*, & \forall j = 1,\ldots,N, \\
\quad W_k \in \mathbb{R}^N, & \forall k = 1,\ldots,m.
\end{cases}$$

4 Solving the model by simulated annealing

4.1 Definitions and notations

Hamming distance Let $x, y \in \{0,1\}^N$, the Hamming distance $d(x,y)$ between x and y is by definition the number of positions for which x and y differ:

$$d(x,y) = \sum_{i=1}^{N} |x_i - y_i|. \tag{14}$$

Neighborhood A neighborhood system on set S is a set collection $N = \{N(s)\}_{s \in S}$ consisting of subsets of S, where for any $s \in S$

- $s \in N(s)$,
- $t \in N(s) \Rightarrow s \in N(t)$,

where $t \in N(s)$ is called neighbor of s, and the neighboring pairs are denoted by $\langle s,t \rangle$.

Neighborhood of order q: $N_q(x) = \{y \in \{0,1\}^N | d(x,y) \leq q\}$.

4.2 Proposed algorithm

The algorithm begins by setting the parameters for a RBFNN with maximum size i.e. $N = N_{max}$. The starting point for our Algorithm 3 is initialized as follows:

- $\forall i \in \{1,\ldots,N_{max}\}\ v_i^0 = 1$.
- u^0 determine by some iterations in k–means.
- $c_i^0 = \frac{\sum_{h=1}^{n} u_{hi}^0 x^h}{\sum_{h=1}^{n} u_{hi}^0}\ \forall i \in \{1,\ldots,N_{max}\}$.
- $\sigma_j^0 = \min\left\{ \left\| c_i^0 - c_j^0 \right\| : c_i^0 \neq c_j^0, i = 1,\ldots,N \right\}, \quad \forall j \in \{1,\ldots,N_{max}\}$.
- $(W^0)^T = [\Phi\Phi^t]^{-1}\Phi\mathbf{Y}$.

Iteration: The Algorithm 3 proceeds iteratively through the following steps:

- We choose $v^{'}$ a neighborhood with order 1 of v randomly follows the uniform distribution by the following steps:

  1. $v^{'} = v$.
  2. $t = rand\ (1, N_{max})$.
  3. $v_t^{'} = \begin{cases} 1, & \text{if } v_t = 0, \\ 0, & \text{if } v_t = 1. \end{cases}$

Algorithm 2 : u Selection method

Require: t, $v^{'}$, u
1: $u^{'} = u$.
2: **if** $v_t^{'} = 0$ **then**
3: **for** $j \in \{1,\ldots,n\}$ **do**
4: **if** $u_{tj} = 1$ **then**
5: $u_{jt}^{'} = 0$,
6: $k = argmin_{i\in\{1,\ldots,N_{max}\}} \left\|c_i - x_j\right\|$,
7: $u_{kt}^{'} = 1$.
8: **end if**
9: **end for**
10: **else**
11: $u^{'} := rand(\{x \in E(v^{'})/d(x,u) = 2\})$.
12: **end if**
13: **return** $u^{'}$.

- We choose $u^{'}$ if $v_t^{'} = 0$ as the orthogonal projection of $(v^{'},u,\sigma,W)$ on S the set of feasible solutions. If not, we make a random change with the uniform law on u to return $(v^{'},u^{'},\sigma,W)$ a feasible solution, by the following formula:

$$u^{'} := \begin{cases} \text{argmin}_{x\in E(v^{'})} d(u,x), & \text{if } v_t^{'} = 0, \\ \text{rand}(\{x \in E(v^{'})/d(x,u) = 2\}), & \text{if } v_t^{'} = 1, \end{cases}$$

where $E(v^{'}) = \{u^{'} \in \{0,1\}^{n\times N_{max}} : \ g_1(u^{'},v^{'}) = 0, \quad g_{2i}(u^{'}) = 0\ ,\ \forall i \in \{1,\ldots,n\}$ and $g_{3j}(u^{'},v^{'}) = 0\ ,\ \forall j \in \{1,\ldots,N_{max}\}\}$. Algorithm 2 explains more about the selection method where

$$c_i = \frac{\sum_{h=1}^{n} u_{hi} x^h}{\sum_{h=1}^{n} u_{hi}}.$$

- We calculate the variance σ with the following expression

$$\sigma_j^{'} = \min\left\{\left\|c_i^{'} - c_j^{'}\right\| : c_i^{'} \neq c_j^{'}, i = 1,\ldots,N\right\}, \ \ \forall j \in \{1,\ldots,N_{max}\},$$

where $c_i^{'} = \frac{\sum_{h=1}^{n} u_{hi}^{'} x^h}{\sum_{h=1}^{n} u_{hi}^{'}}, \quad \forall i \in \{1,\ldots,N_{max}\}$.

- The output layer weights are calculated with the following expression: $(W')^T = \left[\Phi\Phi^t\right]^{-1}\Phi\mathbf{Y}$.

This overarching algorithm optimizes the system parameters (u,v,σ,W) to minimize the objective function F_{eq}, while considering certain constraints and objectives. It operates as follows:

1. **Iterative optimization:** MOSA iterates until convergence, adjusting the temperature parameter T dynamically. This temperature parameter controls the acceptance probability of unfavorable solutions, allowing the algorithm to escape local optima. At each iteration of the algorithm:

Algorithm 3 : Multiple objective simulated annealing

Require: $\lambda_1, \lambda_2, \lambda_3$, T, $(u\ , v\ , \sigma\ , W)$, ΔT, X, Y, $MaxItr$.

1: $(u^*, v^*, \sigma^*, W^*) = (u\ , v\ , \sigma\ , W)$
2: **while** not(end) **do**
3: $k = 1$
4: **while** $k < MaxItr$ **do**
5: $v^{'} = Neighbor(v)$
6: Compute σ, u and Φ
7: $(W')^T = [\Phi\Phi^t]^{-1}\,\Phi\mathbf{Y}$
8: Compute π_1, π_2 and π_3
9: Compute p and ΔF_{eq}
10: **if** $\Delta F_{eq} < 0$ **then**
11: $(u\ , v\ , \sigma\ , W) = (u^{'}, v^{'}, \sigma^{'}, W^{'})$
12: **else**
13: $\alpha = rand(0,1)$
14: **if** $\alpha < p$ **then**
15: $(u\ , v\ , \sigma\ , W) = (u^{'}, v^{'}, \sigma^{'}, W^{'})$
16: **else**
17: $(u\ , v\ , \sigma\ , W) = (u\ , v\ , \sigma\ , W)$
18: **end if**
19: **end if**
20: **if** $F_{eq}(u\ , v\ , \sigma\ , W) < F_{eq}(u^*, v^*, \sigma^*, W^*)$ **then**
21: $(u^*, v^*, \sigma^*, W^*) = (u\ , v\ , \sigma\ , W)$
22: **end if**
23: k=k+1
24: **end while**
25: $T = \alpha(T)$
26: **if** $T < \Delta T$ **then**
27: end (while)
28: **end if**
29: **end while**
30: **return** $(u^*, v^*, \sigma^*, W^*)$.

- A neighborhood v' is randomly chosen from a set of neighbors of v.
- u' is updated based on v'.
- σ' is calculated based on u'.
- Output layer weights $(W')^T$ are recalculated.
- Various objectives (π_1, π_2, π_3) are computed.
- Changes are accepted or rejected based on the change in objective functions and a probability criterion.
- If the new solution is better, it is accepted; otherwise, it might be accepted probabilistically based on the Metropolis criterion.

- The algorithm iterates until a stopping criterion is met, such as reaching a maximum number of iterations.

2. **Objective function evaluation:** At each iteration, the objective function F_{eq} is evaluated. This function represents the system's performance, incorporating both the accuracy of the RBFNN and adherence to constraints.

3. **State transition:** Proposed changes to the system parameters are evaluated based on whether they improve F_{eq}. If accepted, they become the new state. Otherwise, acceptance is probabilistic, depending on the change's impact and the current temperature T.

This algorithm utilizes SA, a probabilistic optimization technique, to optimize the parameters of a RBFNN based on multiple objectives. It iteratively explores the solution space, permitting occasional moves to escape local optima, and gradually diminishes the exploration range over time until convergence or a predefined stopping criterion is satisfied.

5 Implementations and numerical simulation

5.1 Data set experiments

To highlight the benefits of our model and the suggested approach for obtaining an approximate solution, we conducted tests on clustering problems using three datasets sourced from the UCI machine learning repository [25]. These datasets were chosen to provide diverse computing experiences and included Iris, Seeds, Wine, and Pima Indians Diabetes, each representing distinct natures of data. In Table 1, the dataset details are presented, including the number of examples, number of attributes, and class information.

Table 1: Characteristic of the data set used.

Database	Examples	Attributes	Class
Iris	150	4	3
Seed	210	7	3
Wine	178	13	3
Pima Indians Diabetes	768	8	2

The algorithm parameters are also provided in the Table 2:

Table 2: Algorithm parameter setting.

MaxItr	λ_1	λ_2	λ_3	T	$\Delta(T)$	α
20	0.4	0.4	0.2	2	0.25	0.5

In this study, the division of all datasets into training and testing sets was done with a ratio of 50% for training and 50% for testing. We computed the percentage of misclassification for the training and testing sets using the following formulas:

$$\text{A.MT}\ (\%) = \frac{\text{MT} \times 100}{\text{Size of training set}}, \qquad \text{A.MTS}\ (\%) = \frac{\text{MTS} \times 100}{\text{Size of testing set}}. \tag{15}$$

where MT represents the number of misclassified patterns in the training set, and MTS represents the number of misclassified patterns in the testing set. Also, we determine the percentage of classification errors for both the trained and tested sets using the following formulas:

$$\text{A.T}\ (\%) = \frac{\text{correct classified} \times 100}{\text{Size of training set}}, \qquad \text{A.TS}\ (\%) = \frac{\text{correct classified} \times 100}{\text{Size of Testing set}}. \tag{16}$$

5.2 Classification results

Table 3 provides a comprehensive summary of the classification accuracy rate and convergence iterations used from the proposed method on four datasets: Iris, Wine, Seeds, and Pima Indians Diabetes. By analyzing these results, it is clear that higher accuracies were achieved on all datasets with fewer iterations needed for each. Overall, it shows that the proposed method is quite efficient and exhibits successful performance on all datasets.

Table 3: Classification results.

Data set	It	M.T	M.TS	A.MT	A.MTS
Iris	80	1	2	1.33	2.66
Seeds	80	2	2	1.90	1.90
Wine	80	1	2	1.12	2.24
Pima Indians Diabetes	80	0	1	0	0.26

The ROC curve is a graph that shows how well a binary classifier distinguishes between positive and negative cases as its discrimination threshold varies. The Area Under the Curve (AUC) summarizes the ROC curve into a single value, ranging from 0 to 1. An AUC of 1 represents a perfect classifier, while 0.5 indicates random guessing. A higher AUC suggests better classifier performance, making it a useful metric for comparing models in binary classification tasks. In Figure 2, the ROC curve and AUC metric are presented for all datasets, providing a comprehensive evaluation of the model's performance across various data samples. The ROC curve illustrates the trade–off between true positive rate and false positive rate, while the AUC metric quantifies the model's discriminative ability. This analysis offers valuable insights into the classifier's efficacy in distinguishing between different classes within the datasets.

As shown in Table 4, all initializations yield nearly similar optimal numbers for each dataset. For example, for the "Iris" data, our model returns 12 neurons for $N_{\max} = 30$, 10 neurons for $N_{\max} \in \{40, 50, 60\}$, and 11 neurons for $N_{\max} \in \{20, 25\}$. This result is confirmed by Fig 3, where the number of neurons in the hidden layer reduces with increasing number of iterations until stabilization in the suitable architecture. The results for the "Iris" dataset are shown in Table 3 demonstrate the power of our model and approach. Furthermore, the results for the other three datasets "Wine", "Seeds", and "Pima Indians Diabetes", demonstrate the efficiency of our algorithm, as the number of neurons decreases and stabilizes in some iterations.

As shown in Figure 4, the comparison results of classification rates for training data and test data indicate that our method is a good model for solving the generalization problem. We observe that, across all datasets, the two rates are very close to each other. This demonstrates that our method performs well when it comes to addressing issues related to generalized learning.

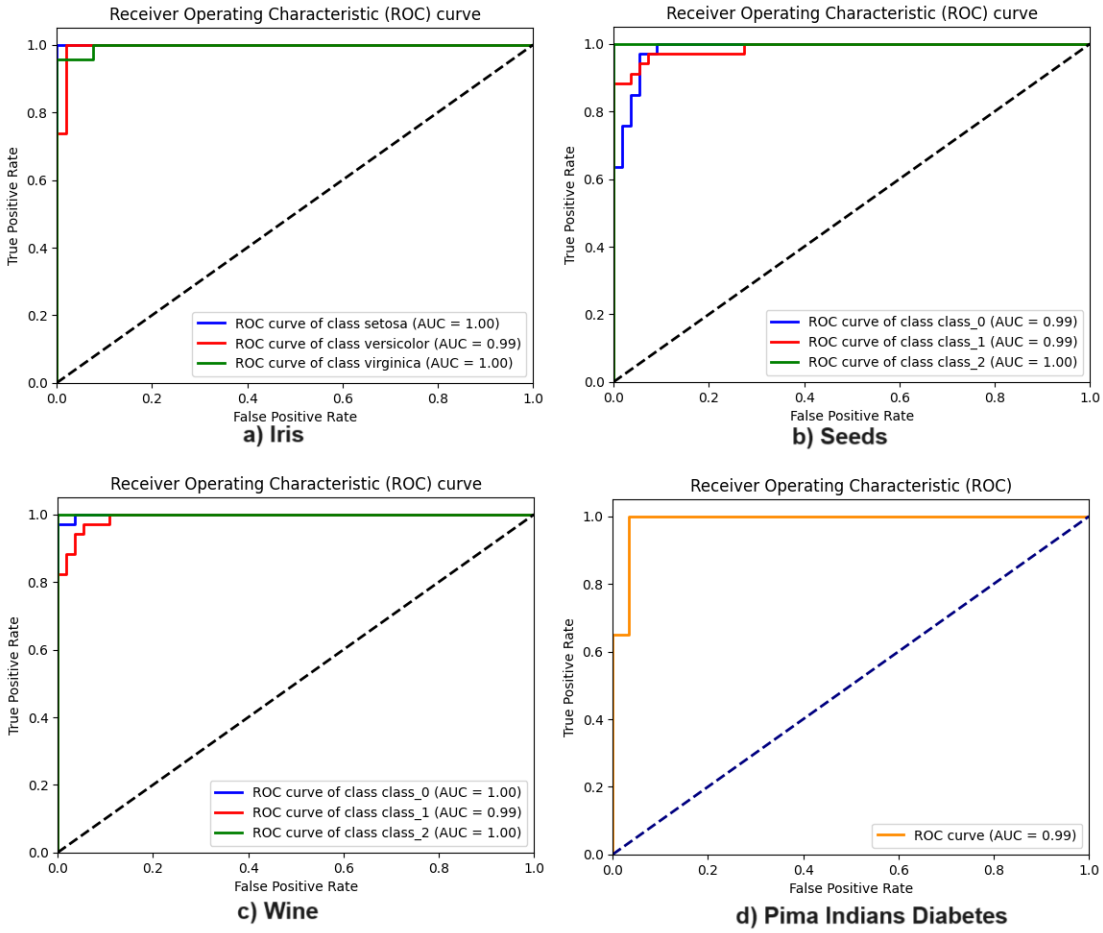

Figure 2: ROC curves and AUC metrics for all datasets.

5.3 Comparison results

5.3.1 Compared to other neural network training algorithms

In this section, we present a comparative analysis of outcomes achieved by various methods on Iris databases, focusing on existing neural network training algorithms documented in the literature. These include Multilayer Perceptron (MLP) trained with Error Backpropagation (EBP), RBFNN trained with EBP, RBFNN trained with hybrid algorithms, and Support Vector Machine (SVM).

By comparing the average classification accuracy rate presented in Table 5, it can be seen that the proposed method outperforms other existing neural network training algorithms in the literature. This can be attributed to the use of specific training algorithms and RBFNN hyperparameter optimization techniques for each given dataset. This advantage is also evidenced by the results of our experiments, which showed better generalization performance compared to other neural network training algorithms.

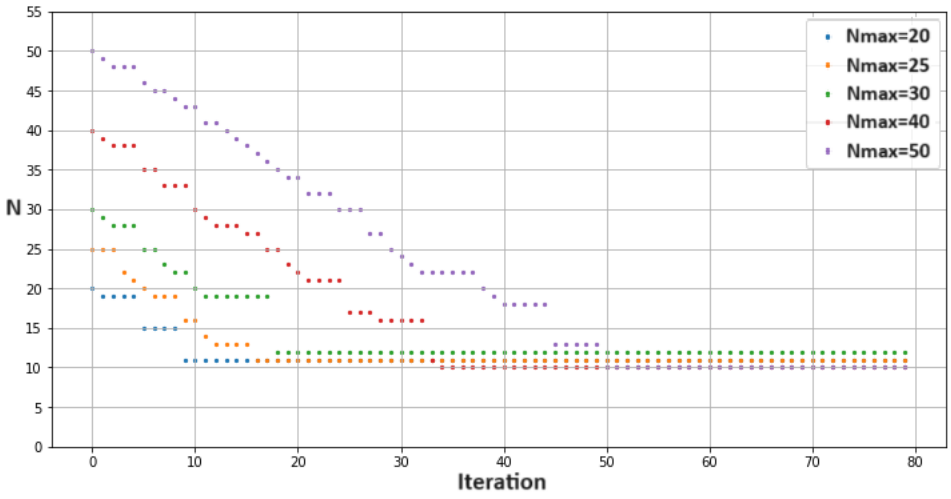

Figure 3: Optimal architecture for the Iris dataset with different initial numbers of neurons.

Table 4: Optimal topological maps sizes of Dataset.

Data set	20	25	30	40	50	60
Iris	11	11	12	10	10	10
Seeds	14	14	15	16	15	16
Wine	7	10	12	11	11	12
Pima Indians Diabetes	2	2	2	2	2	2

5.3.2 Compared to other similar optimization methods

Hyperparameter optimization is a crucial part of machine learning, and there are many methods to accomplish it. We will describe some of them and compare them with our proposed method in this section.

- Grid Search (GR) is an optimization method that allows us to test a series of parameters and compare performance to deduce the best setting.
- Random Search is a stochastic search approach that randomly samples from the parameter space and evaluates the performance of each sample, and then trains such models until the specified resource is exhausted [3].
- Successive Halving (SH) serves as an optimization algorithm that involves randomly selecting a group of hyperparameter configurations, assessing the performance of all configurations, discarding the lower–performing half, and then repeating this process iteratively until only one configuration remains [17].
- Hyperband is an enhancement built upon successive halving algorithms, as put forth by Lisha Li and colleagues in their work [23].

Table 6 reveals that employing default hyperparameter settings in our experiment yields suboptimal model performance, emphasizing the significance of employing hyperparameter optimization methods.

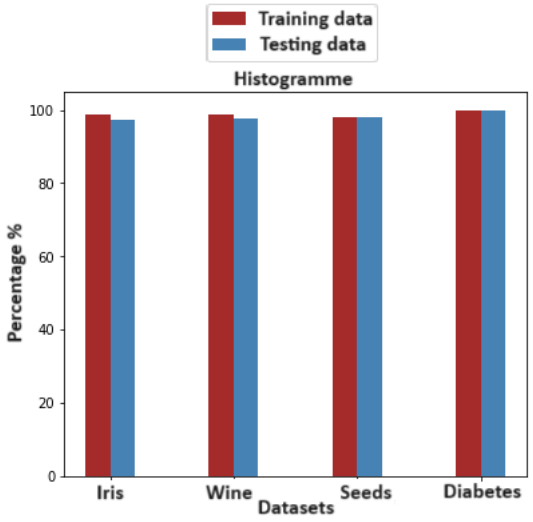

Figure 4: Comparison between classification rates of training and testing data.

Table 5: Comparison results.

Method	N	M.T	M.TS	A.T	A.TS
RBFNN PB	16	2	3	97.30	96.00
RBFNN Hybrid	16	2	3	97.30	96.00
MLPNN EPB	16	2	3	97.30	96.00
SVM	–	3	5	96.00	93.33
P.Method	–	1	2	97.33	98.66

The computational time for GS and RS tends to be notably higher than other HPO methods, despite achieving satisfactory accuracy. Hyperband achieves results more swiftly but at the expense of lower accuracy compared to alternative methods. Our approach not only delivers optimal model performance but also does so efficiently in terms of time. This ensures the identification of optimal hyperparameters and parameters for our models.

6 Conclusion

In this paper, we introduced a novel model for optimizing the architecture and learning process of RBFNN. This model is specifically designed to identify the optimal number of neurons in the hidden layer. Our innovative approach, leveraging multi–objective simulated annealing, enables the determination of optimal parameters and hyperparameters for neural networks. This ensures the development of a well–balanced neural network capable of solving problems without succumbing to overfitting or underfitting, thereby enhancing the generalization capacity of RBFNN. Furthermore, the proposed algorithm for approximating the solution demonstrates efficacy, as evidenced by the obtained results. This suggests that our model can yield robust solutions. Additionally, we plan to explore the application of other opti-

Table 6: Compared to other optimization methods for Iris.

Method	Accuracy(%)	TC(s)
Default HPs	93.33	2.31
Grid Search	96.00	30.00
Random Search	94.66	27.12
Successive halving	96.00	13.51
Hyperband	96.00	14.21
Genetic Algorithm	96.00	15.01
P.Method	**97.33**	**9.71**

mization algorithms to solve our model and extend our technique to address real–world problems using diverse databases.

Acknowledgements

We would like to express our sincere gratitude to the editorial team of JMM for their invaluable guidance and support throughout the preparation of this manuscript. We also extend our heartfelt thanks to the reviewers for their thorough and insightful feedback, which has enhanced the quality of this work.

References

[1] A. Abbassi, *A robust optimization approach for multi–objective linear programming under uncertainty*, J. Math. Model. **11** (2023) 695–708.

[2] T. Baklacioglu, *Predicting the fuel flow rate of commercial aircraft via multilayer perceptron, radial basis function and ANFIS artificial neural networks*, Aeronaut. J. **125** (2021) 453–471.

[3] J. Bergstra, R. Bardenet, Y. Bengio, B. Kegl, *Algorithms for hyper–parameter optimization* , Advances in Neural Information Processing Systems, Curran Associates, 2011.

[4] S.A. Billings, G.L. Zheng, *Radial basis function network configuration using genetic algorithms*, Neural Netw. **8** (1995) 877–890.

[5] D.S. Broomhead, D. Lowe, *Radial Basis Functions, Multi–Variable Functional Interpolation and Adaptive Networks*, Royals Signals and Radar Establishment, Malvern, 1988.

[6] M. Chun–Tao, L. Xiao–Xia, Z. Li–Yong, *Radial basis function neural network based on ant colony optimization*, Int. Conf. Comput. Intell. Secur. Workshops. IEEE, (2007, December) 59–62.

[7] K. Deb, *Multi–Objective Optimization*, Search Methodologies, Springer, Boston, 2014.

[8] M. Dua, R. Gupta, M. Khari, R.G. Crespo, *Biometric iris recognition using radial basis function neural network*, Soft Comput. **23** (2019) 11801–11815.

[9] T. Elansari, M. Ouanan, H. Bourray, Mixed *Radial Basis Function Neural Network Training Using Genetic Algorithm*, Neural Process. Lett. **55** (2023) 10569–10587.

[10] E. Hafezieh, A. Tavakoli, M. Matinfar, *Strategies for disease diagnosis by machine learning techniques*, J. Math. Model. **11** (2023) 441–450.

[11] T. Hatanaka, N. Kondo, K. Uosaki, *Multi–Objective Structure Selection for RBF Networks and Its Application to Nonlinear System Identification*, Multi-Objective Machine Learning, Springer, Berlin, Heidelberg, 2006.

[12] H. Hong, Z. Zhang, A. Guo, L. Shen, H. Sun, Y. Liang, F. Wu, H. Lin, *Radial basis function artificial neural network (RBF ANN) as well as the hybrid method of RBF ANN and grey relational analysis able to well predict trihalomethanes levels in tap water*, J. Hydrol. **591** (2020) 125574.

[13] H.C. Huang, S.K. Lin, *A hybrid metaheuristic embedded system for intelligent vehicles using hypermutated firefly algorithm optimized radial basis function neural network*, IEEE Trans. Ind. Inform. **15** (2018) 1062–1069.

[14] H. Jabbar, R.Z. Khan, *Methods to avoid over–fitting and under–fitting in supervised machine learning (comparative study)*, Computer Science, Communication and Instrumentation Devices **70** (2015) 978–981.

[15] N.E. Joudar, Z. En–Naimani, M. Ettaouil, *Using continuous Hopfield neural network for solving a new optimization architecture model of probabilistic self organizing map*, Neurocomputing **344** (2019) 82–91.

[16] N.B. Karayiannis, *Reformulated radial basis neural networks trained by gradient descent*, IEEE Trans. Neural Netw. **10** (1999) 657–671.

[17] Z. Karnin, T. Koren, O. Somekh, *Almost optimal exploration in multiarmed bandits*, Int. Conf. Mach. Learn. PMLR **28** (2013) 1238–1246.

[18] A. Khodadadian, M. Parvizi, C. Heitzinger, *An adaptive multilevel Monte Carlo algorithm for the stochastic drift-diffusion-Poisson system*, Comput. Methods Appl. Mech. Eng. **368** (2020) 113163.

[19] A. Khodadadian, M. Parvizi, M. Teshnehlab, C. Heitzinger, *Rational design of field–effect sensors using partial differential equations, Bayesian inversion, and artificial neural networks*, Sensors **22** (2022) 4785.

[20] L.I. Kuncheva, *Initializing of an RBF network by a genetic algorithm*, Neurocomputing **14** (1997) 273–288.

[21] R.J. Kuo, L.M. Ho, C.M. Hu, *Integration of self–organizing feature map and K–means algorithm for market segmentation*, Comput. Oper. Res. **29** (2002) 1475–1493.

[22] S. Lee, R.M. Kil, *A Gaussian potential function network with hierarchically self–organizing learning*, Neural Netw. **4** (1991) 207–224.

[23] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, *Hyperband: A novel bandit–based approach to hyperparameter optimization*, J. Mach. Learn. Res. **18** (2017) 6765–6816

[24] Y. Liao, S.C. Fang, H.L.W. Nuttle, *Relaxed conditions for radial–basis function networks to be universal approximators*, Neural Netw. **16** (2003) 1019–1028.

[25] M. Lichman, *UCI machine learning repository*, 2013, http://archive.ics.uci.edu/ml.

[26] W.A. Light, *Some aspects of radial basis function approximation*, Approximation Theory, Spline Functions and Applications, Springer, Dordrecht (1992) 163–190.

[27] A.C. Lorena, L.P. Garcia, J. Lehmann, M.C. Souto, T.K. Ho, *How Complex is your classification problem? A survey on measuring classification complexity*, ACM Comput. Surv. **52** (2019) 1007.

[28] M. Madhiarasan, *Accurate prediction of different forecast horizons wind speed using a recursive radial basis function neural network*, Prot. Control Mod. Power Syst. **5** (2020) 1–9.

[29] C.A. Micchelli, *Interpolation of scattered data: distance matrices and conditionally positive definite functions*, In Approximation Theory, Spline Functions and Applications, Springer, Dordrecht (1984) 143–145.

[30] M.T. Musavi, W. Ahmed, K.H. Chan, K.B. Faris, D. M. Hummels, *On the training of radial basis function classifiers*, Neural Netw. **5** (1992) 595–603.

[31] V. Pesce, S. Silvestrini, M. Lavagna, *Radial basis function neural network aided adaptive extended Kalman filter for spacecraft relative navigation*, Aerosp. Sci. Technol. **96** (2020) 105527.

[32] M.J.D. Powell, *Radial basis functions*, Adv. Numer. Anal. **2** (1992) 105–210.

[33] F.B. Rizaner, A. Rizaner, *Approximate solutions of initial value problems for ordinary differential equations using radial basis function networks*, Neural Process. Lett. **48** (2018) 1063–1071.

[34] S. Shashaani, M. Teshnehlab, A. Khodadadian, M. Parvizi, T. Wick, N. Noii, *Using layer–wise training for road semantic segmentation in autonomous cars*, IEEE Access (2023).

[35] A.F. Sheta, K. De Jong, *Time–series forecasting using GA–tuned radial basis functions*, Inf. Sci. **133** (2001) 221–228.

[36] K.P. Sinaga, M.S. Yang, *Unsupervised K–means clustering algorithm*, IEEE access **8** (2020) 80716–80727.

[37] B. Suman, P. Kumar, *A survey of simulated annealing as a tool for single and multiobjective optimization*, J. Oper. Res. Soc. **57** (2006) 1143–1160.

[38] S. Lawrence, C.L. Giles, *Overfitting and neural networks: conjugate gradient and backpropagation*, Proc. IEEE–INNS–ENNS Int. Joint Conf. Neural Netw. Neural Comput: New Challenges and Perspectives for the New Millennium, **1** (2000) 114–119.

[39] E. Taoufyq, O. Mohammed, B. Hamid, *Multi–objective optimization of radial basis function neural network training using genetic algorithm*, AIP Conf. Proc. AIP Publishing, **3034** (2024).

[40] T. Uhl, *The inverse identification problem and its technical application*, Arch. Appl. Mech. **77** (2007) 325–337.

[41] H. Wang, L. Zhang, K. Yin, H. Luo, J. Li, *Landslide identification using machine learning*, Geosci. Front. **12** (2021) 351–364.

[42] B. Yu, X. He, *Training radial basis function networks with differential evolution*, Proc. IEEE Int. Conf. Granular. Comput. 2006.

[43] H.R. Yousefzadeh, E. Zahiri, A. Heydari, *Ranking the Pareto frontiers of multi–objective optimization problems by a new quasi–Gaussian evaluation measure*, J. Math. Model. **11** (2023) 55–70.

[44] E.J. Zaferani, M. Teshnehlab, A. Khodadadian, C. Heitzinger, M. Vali, N. Noii, T. Wick, *Hyper–parameter optimization of stacked asymmetric auto–encoders for automatic personality traits perception*, Sensors, **22** (2022) 6206.