Neural Computing and Applications
https://doi.org/10.1007/s00521-020-05131-y

ORIGINAL ARTICLE

Memetic algorithms for training feedforward neural networks: an approach based on gravitational search algorithm

Ricardo García-Ródenas[1] · Luis Jimenez Linares[2] · Julio Alberto López-Gómez[2]

Received: 12 July 2019 / Accepted: 16 June 2020

Abstract
The backpropagation (BP) algorithm is a gradient-based algorithm used for training a feedforward neural network (FNN). Despite the fact that BP is still used today when FNNs are trained, it has some disadvantages, including the following: (i) it fails when non-differentiable functions are addressed, (ii) it can become trapped in local minima, and (iii) it has slow convergence. In order to solve some of these problems, metaheuristic algorithms have been used to train FNN. Although they have good exploration skills, they are not as good as gradient-based algorithms at exploitation tasks. The main contribution of this article lies in its application of novel memetic approaches based on the Gravitational Search Algorithm (GSA) and Chaotic Gravitational Search Algorithm (CGSA) algorithms, called respectively Memetic Gravitational Search Algorithm (MGSA) and Memetic Chaotic Gravitational Search Algorithm (MCGSA), to train FNNs in three classical benchmark problems: the XOR problem, the approximation of a continuous function, and classification tasks. The results show that both approaches constitute suitable alternatives for training FNNs, even improving on the performance of other state-of-the-art metaheuristic algorithms such as ParticleSwarm Optimization (PSO), the Genetic Algorithm (GA), the Adaptive Differential Evolution algorithm with Repaired crossover rate (Rcr-JADE), and the Covariance matrix learning and Bimodal distribution parameter setting Differential Evolution (COBIDE) algorithm. Swarm optimization, the genetic algorithm, the adaptive differential evolution algorithm with repaired crossover rate, and the covariance matrix learning and bimodal distribution parameter setting differential evolution algorithm.

Keywords Feedforward neural networks · Memetic algorithms · Gravitational search algorithm · Quasi-Newton methods

Abbreviations

BP	Backpropagation
FNN	Feedforward neural network
MA	Memetic algorithm
GSA	Gravitational search algorithm
CGSA	Chaotic gravitational search algorithm
MGSA	Memetic gravitational search algorithm
MCGSA	Memetic chaotic gravitational search algorithm
qN	Quasi-Newton
PSO	Particle Swarm optimization
GA	Genetic algorithm
DE	Differential evolution
Rcr-JADE	Adaptive differential evolution algorithm with repaired crossover rate
COBIDE	Covariance matrix learning and bimodal distribution parameter setting differential evolution
MSE	Mean square error
SSE	Sum of square error
RMSE	Root mean square error
ANFIS	Adaptive-network-based fuzzy inference system
EEMD	Ensemble empirical mode decomposition
EELM	Enhance extreme learning machine
BFOA	Bacterial foraging optimization algorithm
SBLLM	Sensitivity-based linear learning method
EML	Extreme machine learning
LIBS	Laser-induced breakdown spectroscopy

✉ Julio Alberto López-Gómez
JulioAlberto.Lopez@uclm.es

Ricardo García-Ródenas
Ricardo.Garcia@uclm.es

Luis Jimenez Linares
Luis.Jimenez@uclm.es

[1] Department of Mathematics, University of Castilla la Mancha, Ciudad Real, Spain

[2] Department of Information Technologies and Systems, University of Castilla la Mancha, Ciudad Real, Spain

Published online: 06 July 2020

MLP	Multilayer perceptron
IoT	Internet of things

Neural network training problem

n_j	Number of neurons of the layer j
c	Total number of layers in a FNN
W_j	Weight matrix associated with the connections from layer $j-1$ to layer j
b_j	Threshold vector of the neurons for layer j
x_i	Input vector for the FNN
$x_i^{(j)}$	Activation for the neuron i in layer j
s	Activation function for the FNN
o_i	Activation of the neuron i from the output layer
$\boldsymbol{\omega}$	Parametrization of the FNN, denoted by $\boldsymbol{\omega} = \{W_j, b_j\}_{j=1}^{c}$

Memetic chaotic gravitational search algorithm

N_{pob}	Number of initial solutions in the population
t	Current iteration
T	Maximum number of iterations
M_i^t	Mass of the solution i in iteration t
worst^t	Worst fitness solution in iteration t
best^t	Best fitness solution in iteration t
F_{ij}^t	Gravitational force which acts from j to mass i
F_i^t	Total gravitational force acting on the mass i
M_{pi}^t	Passive gravitational mass of i in iteration t
M_{aj}^t	Active gravitational mass of j in iteration t
R_{ij}	Euclidean distance which separates masses i and j
G_0	Initial value of the gravitational constant
G^t	Gravitational constant at iteration t
G_{chaotic}^t	Gravitational constant at iteration t where a chaotic map is added
α	Learning rate
ϵ	Small value to avoid dividing by zero
a_i^t	Acceleration of mass i at iteration t
v_i^{t+1}	Speed of mass i at iteration $t+1$
$\boldsymbol{d}_k$	Search direction at kth iteration
$\nabla f(\boldsymbol{\omega}_k)$	Vector of first-order partial derivatives of the loss function with respect to the vector $\boldsymbol{\omega}$
H_k	Positive definite matrix
$\nabla^2 f(\boldsymbol{\omega}_k)$	Is the Hessian matrix of f at $\boldsymbol{\omega}_k$

1 Introduction

Neural networks are one of the most popular and successful machine learning techniques. The aim of machine learning techniques is to provide algorithms with the goal of allowing the computer to learn automatically without human intervention, and to adjust actions accordingly. Specifically, FNNs are the best known neural network architecture (see [33]), with the layers interconnected in such a way that the output of ith layer is used as the input of the $i+1$th layer. Most commercial applications which use neural networks implement this kind of architecture, since it has been shown to be suitable for addressing a large class of problems pertaining to pattern recognition or prediction (see [8, 41, 42, 67]). Moreover, an FNN with only one hidden layer is considered a universal approximator (see [31]) as it can approximate any continuous function.

In neural networks, the learning process takes place through the adjustment of the weights and biases which measure the strength of the connection between neurons from different layers. These weights are usually updated in order to minimize the error made by the network, which is formulated using a *loss* function, like mean square error (MSE), sum of square errors (SSE) or root mean square error (RMSE). The process of adjusting the weights and biases for a given network is known as *training*. Therefore, from the mathematical point of view, behind a neural network lies the solution of an unconstrained optimization problem.

In the past, gradient-based techniques were the most widely used for FNN optimization (see [32]). These algorithms use a specialized method for computing the gradient of the loss function with respect to the parameters of the network, known as the *backpropagation* technique. The most representative algorithm of this class is the *steepest descent approach* known as the BP algorithm (see [21]). This algorithm is based on the idea of propagating the errors made by the network to the back layers. This identifies how the error varies with respect to the network parameters and adjusts the weights and biases in order to minimize the loss function.

Training an FNN using a BP algorithm is difficult when the problems are non-differentiable or multimodal. This is because gradient-based algorithms are liable to becoming trapped local minima (see [26]), making their performance highly dependent on the initial values of weights, biases and the chosen hyperparameters of the optimization algorithm. Furthermore, the vanishing gradient problem becomes a major concern when many hidden layers are added to the network. These facts motivate the challenge of finding new approaches and algorithms which outperform the BP in these scenarios. In order to address these problems, metaheuristic algorithms have been successfully applied to training FNNs (see [12, 52, 53]). Metaheuristics improve the exploration capacity of gradient-based algorithms, but they exhibit slow convergence in comparison with gradient-based approaches, which achieve convergence at a (super-)linear rate (see [4]).

In the last few years, MAs have appeared as a new computation paradigm which tries to combine metaheuristic algorithms with one or more local search procedures in order to combine the advantages of both approaches (see [47, 50]). MAs have been successfully applied in different domains such as the train timetabling problems (see [14]) and segmentation of temporal series (see [40]).

Recently, [20] have proposed two memetic algorithms for unconstrained optimization based on the hybridization of the GSA (see [60]) or CGSA (see [44]) with quasi-Newton (qN) search directions. The resulting algorithms are named MGSA and MCGSA. In this previous study, the authors proved these memetic algorithms can be considered as part of the state of the art, since they outperformed metaheuristic algorithms from the state of the art in a wide set of synthetic and real-world global optimization problems. This paper studies the application of both memetic approaches to a challenging machine learning problem, specifically the training of an FNN. Both approaches are particularly suited to training a FNN as they consider a qN algorithm with a superlinear convergence rate and they include a promising evolutionary algorithm, so as not to become trapped in local minima. The main advantage of using a metaheuristic algorithm instead of BP is that metaheuristic algorithms are able to escape from local optima, while BP may be trapped in local minima. However, the exploitation capabilities of BP improve upon those of metaheuristic algorithms significantly. This situation motivates the use of the memetic approaches based on gradient information, which combine the advantages of BP and metaheuristic algorithms. Thus, the main advantage of using a memetic approach is a greater rate of convergence than with metaheuristic algorithms and the ability to find a global optimum of the problem, unlike BP. In order to do this, the implementation of these algorithms has been adapted to this problem and the results obtained are analyzed in order to compare the performance of memetic approaches to that of other metaheuristic algorithms from the state of the art.

Furthermore, [45] addressed the training problem of FNN using GSA, PSO and a hybrid of GSA with PSO. This study replicates their computational experiment in order to assess the performance of MGSA and MCGSA in training FNNs. Furthermore, a statistical comparison of both approaches with metaheuristic algorithms from the state of the art was carried out. To do this, PSO and GA were chosen as the traditional algorithms, while Rcr-JADE (see [24]) and COBIDE (see [69]) were selected as recent proposed algorithms, since these variants of the differential evolution (DE) algorithm have been widely used in recent years and give the best results in rankings (see [57]). The results obtained show MCGSA improves statistically, in terms of convergence and speed, on the results obtained by the metaheuristics from the state of the art when training a FNN.

The rest of the article is structured as follows: Sect. 2 gives a brief review of GSA applications in neural networks; Sect. 3 defines and formalizes the neural network training problem. Then, Sect. 4 describes GSA, CGSA and the memetic algorithms used in this paper. Section 5 shows the results of experiments carried out in this study, and finally, Sect. 6 summarizes the conclusions and further work derived from this article.

2 Related work

The success of metaheuristics and evolutionary algorithms in the field of optimization is well known. They are approximate and non-deterministic optimization methods which incorporate mechanisms to escape from local optima in order to reach the global optimum. The popularity of these methods is due, among other things, to their versatility, simple coding, the few hypotheses to be fulfilled and the fact they do not use derivatives. The applications of these algorithms cover many, if not all, fields of engineering.

Some recent applications are related to hydrology, such as [9] where an adaptive-network-basedfuzzy inference system (ANFIS) model is designed for long-term prediction of discharges by the Manwan Hydropower Plant into the Lancangjiang River, in order to manage and schedule the hydroelectric reservoir. Subsequently, [68] proposed a neural network model coupled with the ensemble empirical mode decomposition (EEMD) designed for medium and long-term forecasting in hydrological time series. Hybrid algorithms have also been applied in this field, for example in [46], where a hybrid firefly algorithm with support vector regression is applied to predict evaporation with the purpose of managing water resources and [72], where an enhance extreme learning machine (EELM) is used to forecast river flow in the Kelantan River (Malaysia).

Regarding other fields of science and engineering, recent relevant applications can also be found. For example, in chemistry, [48] have recently proposed approaches based on neural networks, ANFIS and response surface methods to estimate and optimize the main parameters which affect the yield and cost of biodiesel production, and found that neural networks with radial basis functions give

the maximum biodiesel yield production with the minimum production cost. In another field, [17] presented a survey of computational intelligence techniques applied to address severe flood management, identifying neural networks, soft computing techniques, decision trees, fuzzy logic and hybrid approaches. Finally, other recent applications in the field of power systems are [1] where a hybrid algorithm based on PSO and the bacterial foraging optimization algorithm (BFOA) is applied to power system stabilizer design, and [2] where the same authors used GSA to design a static synchronous series compensator for single and multi-machine power systems.

Metaheuristics and evolutionary algorithms in the area of neural networks have been applied to optimize the FNN components of: (i) *connection weights*, i.e., to find the optimal combination of weights and biases which provides the minimum error while keeping the other components fixed at their initial setting; (ii) *architecture*, i.e., the metaheuristic is used to search for optimum architecture from a compact space of FNN topology; and (iii) *hyper-parameters* such as the learning rate in the BP (remember that the efficiency of these algorithms is highly conditioned by these values). The first two areas have attracted the attention of the research community, with many studies appearing related to these fields. The aim of this section is to review the studies and applications of GSA in the field of neural networks, paying special attention to groups (i) and (ii).

2.1 Using GSA for training neural networks

The complexity of the training problem derives from the topology of the network, since the size of the network, in terms of hidden neurons and hidden layers, leads to a large number of parameters to be optimized. Thus, the number of hidden neurons is assumed to be fixed in this kind of problem, or different sizes are proposed in order to study which network topology provides the best performance. The sizes of the hidden layers are usually set based on the previous experience of the practitioners.

The metaheuristics and their hybridizations have been applied to training problem of FNN to avoid the drawbacks of gradient-based algorithms. Hybrid algorithms can be classified into two main categories: (i) combinations of two algorithms to take advantage of the local and global search capacity of both algorithms. [70, 73] thus proposed a memetic algorithm based on GA and BP, while [75] proposed a memetic PSO with BP algorithm for training FNN, and (ii) combinations of two metaheuristics, with the purpose of obtaining a more robust global search algorithm. A representative example of this trend is [34], where PSO and GA were used to design a hybrid algorithm for recurrent neural network design.

Although there are many studies that discuss this kind of algorithm (see [29, 71]), this section will focus on studies which use the GSA for training neural networks. In 2012, [45] proposed a hybrid PSO with GSA (PSOGSA) for training FNN with one hidden layer. They used this algorithm to test the performance of the network over three classical benchmark problems, showing that PSOGSA outperforms PSO and GSA in terms of convergence speed and avoiding local minima.

Later, [62] proposed a hybrid GSA with GA showing the performance of this algorithm outperforms BP algorithm in approximation function problems and the XOR problem. The idea of hybridizing GSA with GA was subsequently used in [36] for tuning damping controller parameters in a unified power flow controller, giving good results. Moreover, [49] proposed a neuro-fuzzy system, developed using PSO hybridized with GSA, in order to predict the scouring process at pile groups due to waves. Recently, [55] have proposed a two-layer FNN trained with sensitivity-based linear learning method (SBLLM) hybridized with GSA for estimating the band gap of a doped titanium dioxide semiconductor using crystal lattice distortion.

2.2 Using GSA for optimizing network topology

Choosing the optimal number of hidden neurons and of hidden layers is a key point when considering FNNs. It is a complex problem, since it has to be tuned for the problem at hand. That is the reason why many studies propose different sizes for the hidden layer and, later, the network which exhibits the best performance is chosen (see [19]).

Thus, [5] uses an Elman-type recurrent neural network to estimate solar radiation intensity in order to determine the maximum power point tracking in photovoltaic systems. The number of hidden neurons of the proposed neural model is optimized using a hybrid binary PSO algorithm and GSA (BPSO–GSA); the best network topology had six hidden neurons. Later, [3] proposed a two-layer FNN optimized with an SBLLM method for estimating relative cooling power of manganite-based materials for magnetic refrigeration enhancement. The number of epochs and hidden neurons of the network was optimized using GSA, giving an optimized network with 67 hidden layers trained over 4468 epochs. Finally, another recent application of this approach is [56], where the authors propose an extreme machine learning (EML) approach for precise quantitative analysis of laser-induced breakdown spectroscopy (LIBS)

spectra. The number of hidden neurons is optimized using a hybrid of a GSA with EML.

The analysis of these recent studies reveals a growing interest in hybrid algorithms based on GSA to optimize FNN. In this paper, we address the problem of choosing an optimal network topology based on memetic algorithms which is not popular and it constitutes a promising direction of research.

3 The neural network training problem

This section describes and formalizes the optimization problem which arises from the training of a FNN. In a FNN, also known as multilayer perceptron (MLP), the network topology is composed by the input layer, a set of hidden layers and, finally, the output layer. The connections between the neurons from different layers are always forward, and usually, all the neurons of one layer are connected to all neurons from the next layer. A general scheme of a FNN or MLP is shown in Fig. 1. In the figure, n_j is the number of neurons in the jth layer and the parameter c denotes the total number of layers in the architecture.

In every FNN, the connections between the neurons from different layers are associated with a real number, the so-called weight of the connection. The weight of a connection expresses the synaptic power of a given connection, which can be excitable (positive sign) or inhibitory (negative sign). Furthermore, each neuron in the topology has an internal threshold (b). It is used as a comparison factor in order to produce the output of the network and activate or not a neuron in the topology.

There are two main steps in the training process of a FNN. First, the inputs of the network are propagated forward in order to obtain an output. Next, the error between the output produced by the network and the desirable value is computed. Finally, in a second step, the errors are propagated backward, adjusting the weights and thresholds for each neuron in the topology in order to minimize an error or loss function (defined by the designer). This last step will be carried out by using an optimization method.

We consider a canonical FNN with fully connected layers. The first is the input layer, the following layers are hidden layers and the last one is the output layer. The sizes of the layers are n_j for $j = 1, \ldots, c$. Let $W_j \in \mathbb{R}^{n_j \times n_{j-1}}$ be the weight matrix associated with the connections from layer $j-1$ to layer j, and let $b_j \in \mathbb{R}^{n_j}$ be the threshold vector of the neurons from layer j for layers $j = 1, \ldots, c$. The process of propagating the inputs of the network forward is the following:

1. *Computing the activation for the neurons from the input layer*. Given an input vector $x_i \in \mathbf{R}^{n_0}$ for the FNN, the neurons of the input layer only transmit the received signal to the next layer, so the activation of the $x_i^{(0)}$ is computed using Eq. (1).

$$x_i^{(0)} = x_i. \tag{1}$$

2. *Computing the activation for the neurons from the hidden layer*. The neurons from the hidden layer process the information received from the neurons of the input layer, applying the activation function to the weighted sum of the activations received. FNN applies successive transformations to the given input x_i by means of Eq. (2).

$$x_i^{(j)} = s\left(W_j \cdot x_i^{(j-1)} + b_j\right) \in \mathbb{R}^{n_j}, \quad j = 1, \ldots, c-1, \tag{2}$$

where the vector $b_j \in \mathbf{R}^{n_j}$ contains the jth layer parameters (biases) and s is a component-wise nonlinear *activation* function. The activation function of a

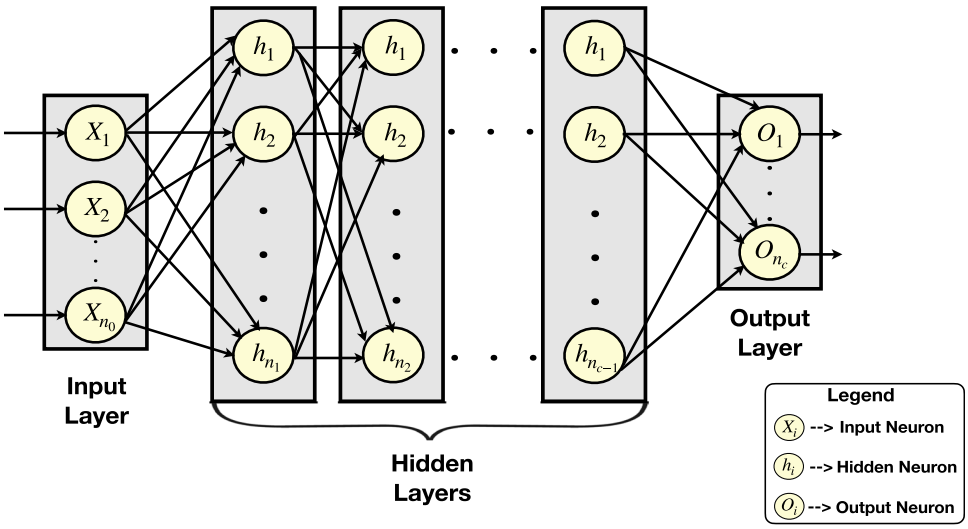

Fig. 1 General architecture of a FNN

neural network includes many alternatives such as sigmoid, hyperbolic tangent, ReLU (rectified linear unit), Leaky ReLU and Softmax. The activation function is necessary to avoid the linearity of the computation since, if a set of neurons are each linked by a process, but without applying a nonlinear activation function; then, the computation of those neurons could have been performed by only one of them. Thus, to provide suitable computing capacity to a neural network it is necessary to establish a nonlinear relationship between the input of each neuron and its output. The most widely used activation functions s of FNNs include the sigmoid function $s(x) = 1/(1+\exp(-x))$ and the ReLU function $s(x) = \max\{0, x\}$ or the hyperbolic tangent $s(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. In this paper, the sigmoid function has been chosen. Hence, Eq. (2) can be rewritten as:

$$x_i^{(j)} = \frac{1}{1 + e^{-(W_j \cdot x_i^{(j-1)} + b_j)}}, \quad j = 1, \ldots, c-1. \tag{3}$$

3. *Computing the activation for the neurons from the output layer*. The last step is to compute the activation of the neurons from the output layer in the same way as it was made in the hidden layer. However, in this case, the activation of the neurons will be the output of the network o_i, as it is shown in Eq. (4).

$$o_i = x_i^{(c)} = s\left(W_c \cdot x_i^{(c-1)} + b_c\right) \in \mathbb{R}^{n_c}, \tag{4}$$

where o_i is the output vector provided by the network for the input vector x_i.

This process in which the neural network obtains the output vector o_i for an input vector x_i, where the parametrization of the network $\boldsymbol{\omega} = \{W_j, b_j\}_{j=1}^{c}$ is known, can be schematically represented by $o_i(x_i, \boldsymbol{\omega})$. Without loss of generality, it can be considered that $\boldsymbol{\omega}$ is a parameter vector $\boldsymbol{\omega} \in \mathbb{R}^n$.

The training problem consists of determining the parameter vector $\boldsymbol{\omega}$ and involves a training dataset $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ and the choice of a loss function ℓ, obtaining the resulting optimization problem:

$$\underset{\boldsymbol{\omega} \in \mathbb{R}^n}{\text{Minimize}} \frac{1}{N} \sum_{i=1}^{N} \ell(o_i(x_i, \boldsymbol{\omega}), y_i). \tag{5}$$

A popular choice for the loss function ℓ is MSE

$$MSE = \ell(o_i(x_i, \boldsymbol{\omega}), y_i) = \|y_i - o_i(x_i, \boldsymbol{\omega})\|_2^2, \tag{6}$$

where $\|\cdot\|_2$ is the Euclidean norm. Hence, Eq. (6) is the objective function of the problem, which can be formalized through Eq. (7).

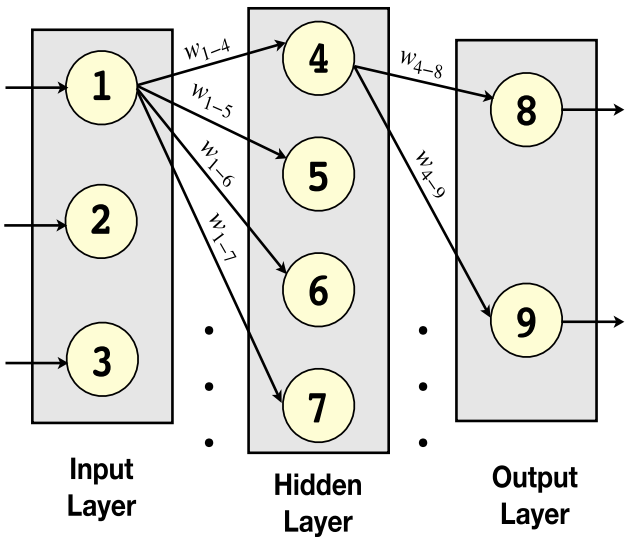

Fig. 2 Example of a FNN network

$$\underset{\boldsymbol{\omega} \in \mathbf{R}^n}{\text{Minimize}} \frac{1}{N} \sum_{i=1}^{N} \|y_i - o_i(x_i, \boldsymbol{\omega})\|_2^2. \tag{7}$$

Therefore, and in accordance with the prior formalization, the training problem of a FNN (without specifying a loss function) can be stated as

$$\underset{\boldsymbol{\omega} \in \mathbb{R}^n}{\text{Minimize}} f(\boldsymbol{\omega}). \tag{8}$$

A small example is shown to illustrate the encoding strategy for the optimization algorithms. Given the fully connected FNN shown in Fig. 2 with three layers where $n_0 = 3, n_1 = 4, n_2 = 2$, a parametrization would be encoded using Eq. (9).

$$W_1 = \begin{bmatrix} w_{1-4} & w_{2-4} & w_{3-4} \\ w_{1-5} & w_{2-5} & w_{3-5} \\ w_{1-6} & w_{2-6} & w_{3-6} \\ w_{1-7} & w_{2-7} & w_{3-7} \end{bmatrix},$$

$$W_2 = \begin{bmatrix} w_{4-8} & w_{5-8} & w_{6-8} & w_{7-8} \\ w_{4-9} & w_{5-9} & w_{6-9} & w_{7-9} \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}, \tag{9}$$

$$\mathbf{b}_3 = \begin{bmatrix} b_8 \\ b_9 \end{bmatrix}.$$

We use the vector $\boldsymbol{\omega}$ to represent the parametrization of the network

$$\boldsymbol{\omega} = [w_{1-4}, w_{1-5}, w_{1-6}, w_{1-7}, \ldots, w_{4-8}, w_{4-9}, \ldots, b_4, b_5, \ldots, b_9]^\top. \tag{10}$$

In population-based metaheuristics, each individual of the population is represented by a vector $\boldsymbol{\omega}$ which includes the weights and biases of the different layers of the network. At the end of the optimization process, the best individual in the population will be used to set the parameters of the neural network.

4 Memetic gravitational search algorithms

This section describes the MGSA and MCGSA training algorithms FNNs. Firstly, the GSA, the CGSA and the qN algorithms are reviewed. Then, the two memetic proposals MGSA and MCGSA are presented.

4.1 Gravitational search algorithm (GSA)

GSA is a bio-inspired algorithm proposed by Rashedi in [60]. It is based on the theory of Newtonian gravity in physics, in such a way that each solution in the search space corresponds to a mass in the space on a universe metaphor. Then, according to the universal law of gravity, heavier masses (solutions with better fitness values) attract the smaller ones (solutions with worst fitness values), ensuring convergence.

The notation that we have used to describe GSA is slightly different from the original formulation, since the notation has been adapted to the neural network training problem. In order not to differ excessively, the notation will be reused, and the indices i and j will be used to refer to population individuals since, in this context, there is no risk of confusion with the hidden layer j or the training data i.

At the start, a population of N_{pob} solutions is randomly initialized, where position and speed values are initialized for each solution. The objective function is then evaluated over all the solutions. The mass of each solution is related to the fitness value in the GSA metaphor. The mass of solution i in iteration t is computed as shown in Eq. (11):

$$M_i^t = \frac{m_i^t}{\sum_{j=1}^{N_{pob}} m_j^t}, \tag{11}$$

where m_i^t is computed through Eq. (12).

$$m_i^t = \frac{f(\boldsymbol{\omega}_i^t) - \text{worst}^t}{\text{best}^t - \text{worst}^t}, \tag{12}$$

where worst^t is the solution with the worst fitness value in iteration t and best^t is the solution which has the best fitness value in iteration t.

Next, the gravitational force acting from each individual to the rest is computed through Newton's law of gravitation, shown in Eq. (13).

$$\mathbf{F}_{ij}^t = G^t \cdot \frac{M_{\mathbf{p}i}^t \cdot M_{\mathbf{a}j}^t}{R_{ij}^t + \epsilon} \cdot \left(\boldsymbol{\omega}_j^t - \boldsymbol{\omega}_i^t\right), \tag{13}$$

where $\mathbf{F}_{ij}$ notes the gravitational force which acts on mass i from mass j. $M_{\mathbf{p}i}$ corresponds to the passive gravitational mass of i mass, while $M_{\mathbf{a}j}$ is the active gravitational mass of j. Note that in GSA, $M_{ai} = M_{pi} = M_i$. R_{ij} is the Euclidean distance that separates the masses i and j. R is used in GSA instead of R^2 because it provides better results (see [60]). ϵ parameter is a small positive constant used to avoid division by zero. Moreover, G^t is the gravitational constant to compute the force. It is initialized to a higher value at the beginning of the algorithm and will be reduced over iterations, according to the following equation $G^t = G_0 \cdot e^{\left(-\alpha \frac{t}{T}\right)}$, where G_0 is the initial value of G, α is the so-called *learning rate*, t is the current iteration and T is the maximum number of iterations allowed. G^t is a exponential function which manages the balance between exploration and exploitation during the execution of the algorithm. At first, high G values give the algorithm greater exploration capability. Subsequently, this value decreases according to the expression shown above, giving the algorithm a greater exploitation capacity. Then, to calculate the total force exerted by one mass of space on another, the random weighted sum of the forces exerted on a concrete mass by the rest of space objects is made. The introduction of the random component in this calculation provides the algorithm with a stochastic behavior. The calculus of total force is shown in Eq. (14).

$$\mathbf{F}_i^t = \sum_{\substack{j=1 \\ j \neq i}}^{N} rand_j \cdot \mathbf{F}_{ij}^t, \tag{14}$$

where $rand_j$ is a single uniformly distributed random number in the interval (0, 1). In addition to the gravitational constant, the GSA has an additional mechanism to control the balance between exploration and exploitation. It is the *Kbest* function in which only the k best objects apply their force to the rest at iteration t. This is a linear function which depends on the iteration t and its value is decreased linearly according to the number of iterations t. It means at first of the algorithm, all the objects exert their force to rest, enhancing exploration, and later, only the *Kbest* objects will apply their force, enhancing exploitation. Thus, the summation in (14) is taken as $j \in Kbest^t - \{i\}$.

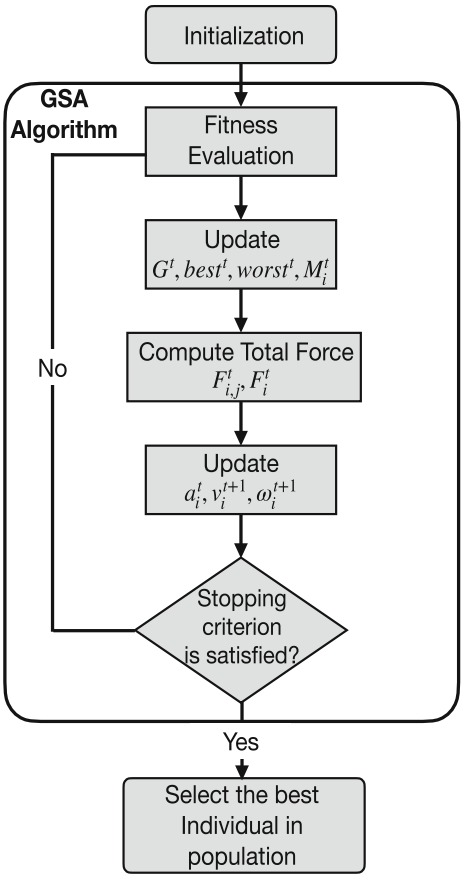

Fig. 3 Flowchart of GSA

Finally, Newton's second law is applied to compute the existing acceleration of each mass, which is computed by $\mathbf{a}_i^t = \frac{\mathbf{F}_i^t}{M_i^t}$ at the end of each iteration. Then, their position and speed will be updated based on the total force. These calculations are expressed as $\mathbf{v}_i^{t+1} = rand_i \cdot \mathbf{v}_i^t + \mathbf{a}_i^t$ and $\boldsymbol{\omega}_i^{t+1} = \boldsymbol{\omega}_i^t + \mathbf{v}_i^{t+1}$. According to these formulae, GSA ensures the movement of the particles toward the particles with greater mass, performing the exploitation when the heavier particles move slowly. Briefly, a flowchart of the GSA is shown in Fig. 3.

4.2 Chaotic gravitational search algorithm (CGSA)

CGSA is a variant of the GSA which appeared in [44] aimed at dealing with the problem of slow convergence inherent to GSA and improving the exploration/exploitation trade-off.

In the GSA, the exploration–exploitation trade-off is controlled by the two functions G^t and $Kbest^t$. On the one hand, $Kbest^t$ encourages the exploration step of the algorithm, allowing all the masses to exert their strength on the rest. This function is decreased using a linear function, and therefore, as the number of iterations increases, the number of masses which exert their force on the rest decreases, focusing the search on the most promising environments found so far. $Kbest^t$ thus encourages the exploitation stage toward the end.

On the other hand, the gravitational constant G^t also controls the trade-off between exploration and exploitation. Initially, G^t values encourage exploration, increasing the force exerted by a mass i on the rest. This value is later decreased using a nonlinear function, in such a way that at the end of the algorithm, the slow movement of heavier agents (thanks to the low values of G^t) encourages the exploitation skills of the GSA, that is, the ability to approximate a local minimum when a promising search environment has been found.

In [44], different chaotic maps are added to G^t in order to perturb the behavior of G^t. In the original GSA, G^t is constantly decreased over the iterations, so the algorithm either explores or exploits. Adding a chaotic map to the gravitational constant will chaotically change the value of G^t while decreasing it during iterations, giving exploration and exploitation at the beginning and at the end of the algorithm. This approach improves significantly the performance of the original GSA, showing that the sinusoidal chaotic map is the most suitable for GSA. A graphical example of introducing a chaotic sinusoidal map into G^t is shown in Fig. 4. The flowchart of CGSA is identical to GSA flowchart, the only difference being if the gravitational constant is replaced by G_{chaotic}^t, and the chaotic GSA appears. CGSA with the sinusoidal map is the CGSA used in the experimental section.

4.3 Gradient-based approaches

In unconstrained optimization tasks, such as the training problem of FNN, *descent direction methods* are widely used. These methods consider the search direction $\boldsymbol{d}_k$ at the kth iteration, making a movement in the direction $\boldsymbol{d}_k$, since these methods guarantee a decrease in the loss function through small movements.

In order to analyze the descent property of these methods, the following function of one variable is defined

$$\phi(\alpha) = f(\boldsymbol{\omega}_k + \alpha \boldsymbol{d}_k). \tag{15}$$

The function ϕ evaluates the progress of the loss function in the direction $\mathbf{d}_k$. If we require the sign of the derivative of the function ϕ in the point $\alpha = 0$ to be negative, to ensure descent for sufficiently small values of α, the search direction must satisfy:

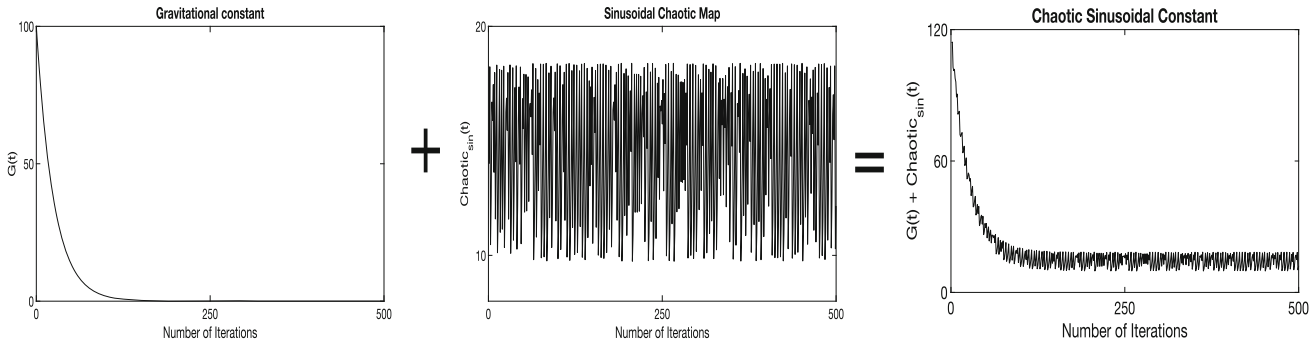

Fig. 4 Obtaining a chaotic sinusoidal gravitational constant

$$\phi'(0) = \nabla f(\boldsymbol{\omega}_k)^\top \boldsymbol{d}_k < 0, \tag{16}$$

where $\nabla f(\boldsymbol{\omega}_k)$ is the vector of first-order partial derivatives of the loss function with respect to the vector $\boldsymbol{\omega}$, i.e., the gradient of the loss function. Condition (16) guarantees that the method is of descent and it allows us to consider weight changes as

$$\Delta\boldsymbol{\omega}_k = \alpha_k \boldsymbol{d}_k, \tag{17}$$

where α_k is small enough to give a decrease in the loss function.

The canonical example is the steepest descent method, which is defined by the choice $\boldsymbol{d}_k = -\nabla f(\boldsymbol{\omega}_k)$ and it satisfies the condition (16)

$$\nabla f(\boldsymbol{\omega}_k)^\top \boldsymbol{d}_k, = -\nabla f(\boldsymbol{\omega}_k)^\top \nabla f(\boldsymbol{\omega}_k) = -\|\nabla f(\boldsymbol{\omega}_k)\|^2 \leq 0. \tag{18}$$

The steepest descent method is a first-order gradient descent algorithm and it is known as BP for training FNN. The numerical scheme leads to:

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \Delta\boldsymbol{\omega}_k = \boldsymbol{\omega}_k - \alpha_k \nabla f(\boldsymbol{\omega}_k), \tag{19}$$

where α_k is the learning rate. The main feature of BP is the gradient calculation specialization for the structure of the problem at hand, a method known as the backpropagation algorithm, where the error obtained at the output layer is propagated backward to the hidden layers.

Several numerical optimization methods appear when certain gradient transformations are applied. If $\mathbf{d}_k$ is a descent direction, the transformation

$$\widehat{\mathbf{d}}_k = -H_k \nabla f(\boldsymbol{\omega}_k), \tag{20}$$

where H_k is a positive definite matrix means the direction $\widehat{\mathbf{d}}_k$ is still a descent direction. The most notable example is *Newton's method* which takes $H_k = [\nabla^2 f(\boldsymbol{\omega}_k)]^{-1}$, where $\nabla^2 f(\boldsymbol{\omega}_k)$ is the Hessian matrix of f at $\boldsymbol{\omega}_k$. Nevertheless, computing $\nabla^2 f(\boldsymbol{\omega}_k)$ implies a high computational cost. In this regard, qN method uses the observed behavior of $f(\boldsymbol{\omega})$ and $\nabla f(\boldsymbol{\omega})$ to compute the approximation H_k to the inverse of the Hessian matrix. This property is what gives the algorithm a quadratic convergence rate, despite considering this approximation.

Furthermore, qN methods are capable of improving the current loss function value. To do that, they need to carry out a precise enough line search

$$\alpha_k := \underset{\alpha \geq 0}{\text{Arg minimize}}\, f(\boldsymbol{\omega}_k + \alpha \widehat{\mathbf{d}}_k), \tag{21}$$

to determine a suitable step length. When the approximate solution α_k of the one-dimensional problem (21) is reached, a new point $\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \alpha_k \widehat{\mathbf{d}}_k$ is obtained with which to update the Hessian matrix.

One of the most widely used Hessian updating methods is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) formula. There is growing evidence that BFGS is the best current update formula for use in unconstrained optimization (see [10]). In problems where the number of parameters is large, such as deep neural networks, a number of strategies have been proposed to address the situation, including limited memory BFGS methods (L-BFGS) (see [39, 51]). Thus, BFGS is considered to be the most effective of all quasi-Newton updating formulae. It approximates the inverse Hessian matrix $[\nabla^2 f(\boldsymbol{\omega}_{k+1})]^{-1}$ by the following expression:

$$H_{k+1} = H_k + \frac{\left(\mathbf{s}_k^\top \mathbf{q}_k + \mathbf{q}_k^\top H_k \mathbf{q}_k\right)\left(\mathbf{s}_k \mathbf{s}_k^\top\right)}{\left(\mathbf{s}_k^\top \mathbf{q}_k\right)^2} - \frac{H_k \mathbf{q}_k \mathbf{s}_k^\top + \mathbf{s}_k \mathbf{q}_k^\top H_k}{\mathbf{s}_k^\top \mathbf{q}_k}, \tag{22}$$

where $^\top$ indicates the transpose and

$$\mathbf{s}_k = \boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k, \tag{23}$$

$$\mathbf{q}_k = \nabla f(\boldsymbol{\omega}_{k+1}) - \nabla f(\boldsymbol{\omega}_k). \tag{24}$$

The procedure is then repeated from the point $\boldsymbol{\omega}_{k+1}$. The initial array H_0 can be adjusted to any symmetric positive definite array, e.g., the identity array. Each iteration can be accomplished at a cost of arithmetic operations $O(n^2)$ (plus the cost of function and gradient evaluations) where n is the number of decision variables. Storage and computing

requirements increase in proportion to n^2 and become impractical for larger n. Many approaches have been presented to deal with this drawback, including quasi-Newton limited memory methods.

4.4 Memetic algorithms based on GSA and CGSA

The proposed memetic algorithms have two main components: (i) an evolutionary framework or a metaheuristic algorithm and (ii) one or more local search methods which will be introduced into approach (i). The two memetic algorithms are proposed according to the following choices: (a) $MGSA = GSA$ for $(i) + qN$ for (ii) and (b) $MCGSA = CGSA$ for (i) + qN for (ii). Both approaches share the local search and differ at the metaheuristic stage, as they are, respectively, GSA and CGSA.

The design of both memetic algorithms is based on the characterization of local minima. The basis of these algorithms is that they are able to detect that a promising neighborhood has been found to trigger the qN method. In this way, adding mechanisms to characterize local minima is a crucial stage in the design of these memetic algorithms. This paper characterizes the concept of a promising neighborhood of an unexplored local minimum using two rules: (i) it contains a solution which has a better fitness value than the best found so far and (ii) it is located far from the current best solution. These rules are formally expressed below:

Rule 1

$$f_*^{t-1} - \text{best}^t > \varepsilon$$

Rule 2

$$\|\boldsymbol{\omega}_{i^*}^t - \boldsymbol{\omega}_*^{t-1}\| > \gamma.$$

In Rule 1, f_*^{t-1} is the best function objective value obtained thus far, $best^t$ is the fitness value of the best solution in iteration t and ε is a parameter which measures whether the new best point is better than the best found so far.

In Rule 2, $\boldsymbol{\omega}_{i^*}^t$ is the best point in the population at iteration t, $\boldsymbol{\omega}_*^{t-1}$ is the best point obtained so far and γ is a parameter which measures whether the new best point is far enough from a neighborhood of $\boldsymbol{\omega}_*^{t-1}$.

Rule 1 states that the point $\boldsymbol{\omega}_{i^*}^t$ reached in the current iteration t is better than the previous points analyzed, and **Rule 2** constrains point $\boldsymbol{\omega}_{i^*}^t$ not to belong to a neighborhood of $\boldsymbol{\omega}_*^{t-1}$, and so, it is suitable to perform an exploitation stage to this new search environment. Parameters $\varepsilon \geq 0$ and $\gamma > 0$ are required to be set in order to detect a promising neighborhood.

However, the detection of a promising region is not the only consideration taken into account by the algorithm, which also looks at where and when the local search method is triggered and the strength with which the local search method will be applied. Fulfillment of both rules (i) and (ii) will trigger the qN method at the end of each GSA or CGSA iteration starting from the best current solution $\boldsymbol{\omega}_{i^*}^t$. Finally, the intensity of the local search is based on a tolerance parameter. The tolerance parameter `tol` represents a lower bound on the size of a step or a decreasing magnitude on the loss function. If the algorithm attempts to take a step that is smaller than the tolerance value, or the improvement is not significant, the iterations of qN end. Thus, the stopping criterion used in the quasi-Newton method is

$$\|\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k\| < \texttt{tol} \vee |f(\boldsymbol{\omega}_{k+1}) - f(\boldsymbol{\omega}_k)| < \texttt{tol}. \quad (25)$$

The `tol` parameter manages the intensity of the local search. At first, it is convenient to explore the neighborhood sparsely, increasing its intensity as the algorithm progresses. For this reason, the parameter `tol` is initialized to $\texttt{tol} = 0.01$. Then it decreases to $\texttt{tol} = \texttt{tol}/10$ each time the quasi-Newton method is performed.

It should be noted that, although the qN search method is used in this paper, any other conventional algorithm can be chosen as a local search component for the memetic algorithm. Specifically, specialized algorithms for solving the neural network training problem, such as Rprop (see [61]), Quickpro (see [15]) or Levenberg–Marquadt (see [66]) algorithms, may be chosen. These algorithms allow specialized gradient computation and have a linear convergence rate. This paper uses the qN method due to its (super-)linear convergence rate.

Finally, in order to summarize the design of the memetic algorithms, Fig. 5 shows a flowchart of both MGSA and MCGSA. In the flowchart, it is possible to see how the metaheuristic algorithm is run, and when a promising area is found (satisfying rules 1 and 2), the qN method is triggered, starting from the best current point. The point obtained by the qN method will replace the best population point, updating not only its position but also its speed. The structure of this memetic algorithm also allows the use of qN method to be extended to non-differentiable problems. If the memetic algorithm achieves a non-differentiable point, then the local search step carried out by qN will be a null stage. However, the memetic algorithm escapes from this problematic point thanks to its evolutionary framework.

5 Experimental results

This section will describe the experiments carried out in this paper in order to study the performance of memetic algorithms based on GSA for training FNN. This was done

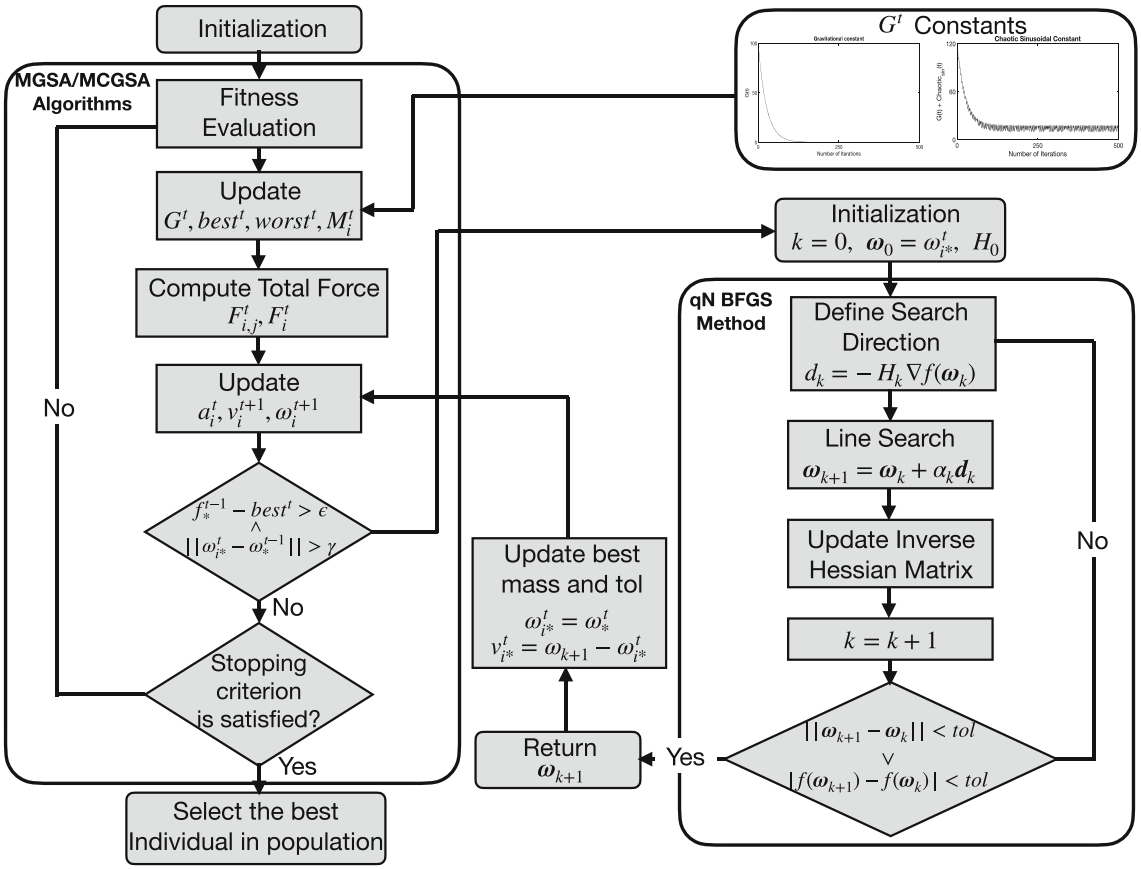

Fig. 5 Flowchart of MGSA and MCGSA algorithms

by replicating the experiments conducted in [45] under the same conditions. These experiments are intended to study if the use of MAs for training FNNs can be a suitable choice in comparison with the current state-of-the-art metaheuristic algorithms. In [45], a hybrid of PSO and GSA was used to train a FNN for three different benchmark problems, showing that the proposed hybrid algorithm outperforms individual GSA and individual PSO.

These authors chose a sample of three classical, well-known benchmark problems where neural networks are used: The first is a three-bit parity (XOR) problem, the second is a function approximation problem and the last one is a classification problem. These problems have been chosen because they are classic benchmarks widely used by the scientific community when a new method for training FNN is designed. Furthermore, this set of problems covers the spectrum of problems that FNN solves: function approximation, binary classification and multi-classification. Another advantage of choosing these problems is the possibility of replicating the experiments of [45] under the same conditions, in order to compare the results obtained with those obtained by the hybrid GSA of these authors. Currently, an alternative to classic benchmarks problems is training deep neural networks as test problems. In these problems, backpropagation has been proved to be inefficient for networks with two or more hidden layers, due to the vanishing gradient problem, poor generalization and the possibility of becoming trapped in a local optimum (see [23, 37, 38]). In this scenario, the challenge is to find alternative methods to deal with these problems. However, the main focus of this paper is on testing hybrid methods to train FNN.

To solve these problems, different algorithms, both classical and from the state of the art of metaheuristics, were chosen to train the FNNs. These are the following: the GSA proposed in [60], the CGSA with a sinusoidal map proposed in [44], and the MGSA and MCGSA described in Sect. 4. These memetic algorithms have been run using, in all trials, a basic configuration ($\varepsilon = 0, \gamma = 1, \texttt{tol} = 0.01$) in order to ensure that the performance of these proposed algorithms is robust with respect to their parametrization, and the parameter tuning is not an inconvenience in their application. The choice of ε and γ means that the qN method is applied each time a mass of the population finds a better solution, especially when it manages to escape from a local minimum. Of course, an optimal tuning will guarantee better results. Also, the PSO algorithm implemented by the *particleswarm* MATLAB function was used

as in [34, 45, 75]. The genetic algorithm implemented in MATLAB by the *ga* function has also been considered in this study in the same way as [34, 62, 65, 73]. Both PSO and GA have been chosen since they are classical metaheuristic algorithms used to train FNNs. Furthermore, in order to compare the memetic algorithms with the state of the art, two algorithms based on the differential evolution (DE) algorithm have been considered in this experiment (see [57]): The first is Rcr-JADE (see [24]). This algorithm modifies the adaptation rule used in the crossover. Also, the adaptation methods used in the control parameters of Rcr-JADE have also been applied in other versions of DE (see [25, 27, 64]). The second algorithm is COBIDE (see [69]). Their authors proposed a new DE variant based on covariance matrix learning and bimodal distribution parameter setting. Both variants of the DE algorithm were chosen since their codes could be publicly obtained and they were in the first positions of the ranking made in [57], where Rcr-JADE was the best algorithm over a set of thirty algorithms in twenty two real-world problems when 50,000 and 100,000 function evaluations were allowed, while the COBIDE algorithm was the best when 150,000 function evaluations were allowed. These experiments have been implemented using the MATLAB programming language, specifically MATLAB 2017b version, and run on a server equipped with a 4.00 gigahertz AMD FX-8370 Eight-Core-Processor.

In order to assess the performance of the algorithms involved in the computational experiments, the goodness of fit reached in the training process through MSE and the computational cost, given as the number of function evaluations, have been used (see [35]). Also, in [54], RMSE is used to assess the performance of the model in many classifiers applied to predict the travel modes of passengers. However, in the case of classification problems, alternative validation measures, such as accuracy, ROC curve or F1 score, are also widely used on training and test problems (see [6, 16, 28]). This paper focuses on the performance of the optimization algorithms employed, and this is the main reason why metrics based on the error measured for by the algorithm in the objective function and the number of function evaluations have been chosen as performance metrics (see [57, 58]).

These three problems are then discussed, with an emphasis on the performance of the memetic algorithms used in this paper, and a statistical comparison is made with the selected algorithms from the state of the art.

5.1 The *m*-bit parity problem (XOR problem)

This is a very famous nonlinear benchmark problem widely used in the optimization of neural networks (see [7, 22, 65]). The definition of this problem is the following:

Table 1 Three-bit parity problem (XOR problem $m = 3$)

Input			Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

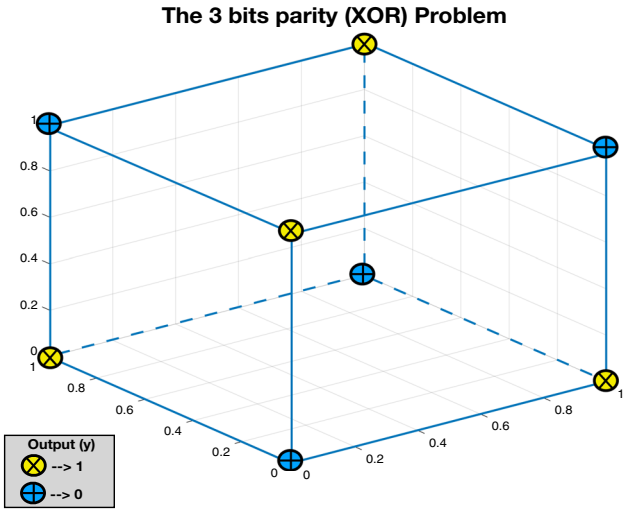

Fig. 6 Landscape of XOR problem with $m = 3$ bits

Given a bit string or input vector of "0s" and "1s" with length m, the desirable output will be 1 if the input vector contains an odd number of "1s" and the output will be 0 if the input vector includes an even number of "1s." Table 1 shows the input vectors and the corresponding outputs for the case $m = 3$ bits.

The main feature of the XOR problem, which makes it a widely used benchmark problem, is that it is not linearly separable. Hence, this problem cannot be solved using a perceptron (a neural network without hidden layers). This feature adds complexity to the problem. Figure 6 shows an illustration to prove this in the case of $m = 3$ bits.

To solve the XOR problem, a feedforward neural network with topology 3-S-1 has been chosen (where S is the number of neurons in the hidden layer). Thus, the input layer has a total of three neurons, according to the length of the input vector in the case of $m = 3$ bits, the hidden layer will have a variable number of neurons which will generate different topologies to be tested, and finally, the output layer has only one neuron which will be 1 or 0, the possible outputs of the XOR problem. The following topologies have also been trained in order to test the performance of the algorithms when the number of parameters increases: $S = \{5, 6, 7, 8, 9, 10, 11, 13, 15, 20, 30\}$. These

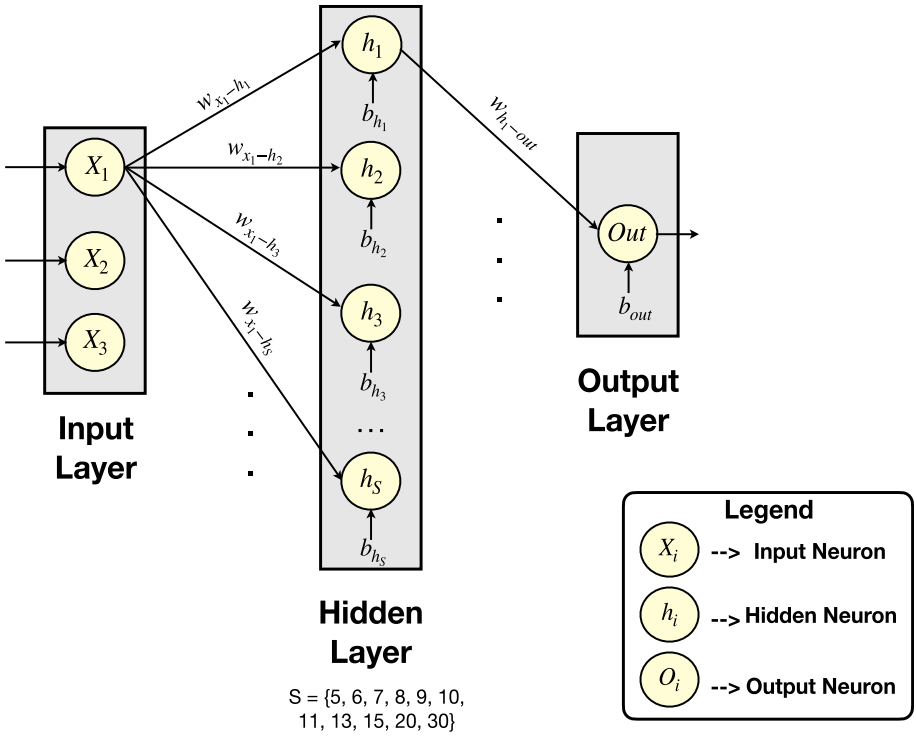

Fig. 7 Graphical representation of the FNN proposed for the XOR problem

network topologies generate a set of optimization problems with a number of dimensions (n) which is easy to compute. For a feedforward neural network with one hidden layer, the number of parameters to be optimized is equal to: $n = n_w + n_b$, where n_w is the number of weights in the neural network and n_b the biases of the neurons. This formula can be extended to the following: $n = n_i \cdot n_s + n_s + n_s \cdot n_o + n_o$, which when simplified is equal to $n = n_s(n_i + 1) + n_o(n_s + 1)$, where n_i is the number of input neurons, n_s the number of hidden neurons and n_o the number of output neurons. The set S of different network topologies therefore generates the following size of problems $n = \{26, 31, 36, 41, 46, 51, 56, 66, 76, 101, 151\}$. Figure 7 shows an illustration of the network topology used to solve the XOR problem.

The setup of the experiment is as follows: each algorithm was run 30 times for each network topology, using a population of 30 individuals ($N_{\text{Pob}} = 30$) where 500 training iterations are allowed. This is the only difference with respect to [45] since these authors carried out a more intensive computational experiment, using a population of 50 individuals over 500 iterations. This paper uses 15,000 evaluations of the function, as solutions are required in a brief time period when addressing real-world problems. In the computational experiments presented in this paper, only 60% of function evaluations are performed in comparison with the experiments carried out in [45].

Tables 2 and 3 display the results of this experiment. Average, median, standard deviation and best values of mean square error (MSE) averaged over the thirty runs have been reported. In addition, the nonparametric statistical test called the Wilcoxon rank-sum test (see [11]) was performed to determine whether the mean performance between algorithms is statistically significant. This test is very suitable when the sample size is small and furthermore does not require that the distribution of the data fulfill any hypothesis, in contrast to parametric tests (see [18]). The statistical significance is calculated by using the $p-$values that are also shown in Tables 2 and 3. The MCGSA has been used as baseline. It should be noted that a $p-$value less than 0.05 means that there are significant differences between the algorithm and the baseline algorithm. Thus, $h = 1$ indicates there is a significant difference in the performance of two algorithms in favor of MCGSA, $h = -1$ implies that the performance of the other algorithm is statistically better, $h = 0$ means that there are no significant differences in the performance of both algorithms, and finally, NA means the test is not applicable, since an algorithm cannot be compared to oneself.

The numerical results show that the use of the proposed memetic algorithms to train a feedforward neural network for the XOR problem improves significantly the results of the GSA family of algorithms and the selected algorithms

Table 2 Statistical results of MSE in a 3-bit parity (XOR) problem (I)

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE	*p* value	*h*
5	MCGSA	6.16E−02	6.25E−02	4.23E−02	2.39E−14	NA	NA
	GSA	1.22E−01	1.25E−01	5.38E−02	1.03E−05	0.00E+00	1
	CGSA	6.58E−02	6.25E−02	6.44E−02	8.61E−14	0.00E+00	1
	MGSA	9.53E−02	1.10E−01	5.40E−02	1.83E−11	0.00E+00	1
	PSO	2.01E+01	2.26E−01	7.72E+01	6.53E−02	0.00E+00	1
	GA	4.36E−02	3.32E−02	3.79E−02	8.34E−04	0.00E+00	− 1
	Rcr-JADE	9.23E−02	8.84E−02	6.17E−02	0.00E+00	0.00E+00	1
	COBIDE	4.63E+02	4.94E+02	1.99E+02	1.03E+02	0.00E+00	1
6	MCGSA	2.37E−02	6.34E−07	3.25E−02	1.17E−13	NA	NA
	GSA	1.42E−01	1.25E−01	1.06E−01	1.16E−05	0.00E+00	1
	CGSA	6.52E−02	6.25E−02	5.87E−02	7.08E−15	0.00E+00	1
	MGSA	4.57E−02	6.25E−02	4.44E−02	3.98E−09	1.59E−96	1
	PSO	5.43E+01	2.26E−01	2.47E+02	1.35E−04	0.00E+00	1
	GA	3.94E−02	2.35E−02	3.47E−02	2.52E−04	0.00E+00	1
	Rcr-JADE	8.71E−02	8.11E−02	6.33E−02	2.46E−29	0.00E+00	1
	COBIDE	5.97E+02	5.69E+02	2.69E+02	1.59E+02	0.00E+00	1
7	MCGSA	2.78E−02	1.26E−06	3.40E−02	3.20E−13	NA	NA
	GSA	1.05E−01	1.09E−01	1.01E−01	1.28E−04	0.00E+00	1
	CGSA	3.17E−02	2.04E−05	4.48E−02	1.54E−14	0.00E+00	1
	MGSA	5.71E−02	3.47E−02	6.82E−02	5.82E−11	0.00E+00	1
	PSO	7.45E+02	2.50E−01	3.13E+03	1.28E−02	0.00E+00	1
	GA	3.55E−02	2.20E−02	3.49E−02	2.00E−03	0.00E+00	1
	Rcr-JADE	8.71E−02	8.11E−02	6.33E−02	2.46E−29	0.00E+00	1
	COBIDE	5.97E+02	5.69E+02	2.69E+02	1.59E+02	0.00E+00	1
8	MCGSA	1.60E−02	9.27E−09	2.76E−02	3.58E15	NA	NA
	GSA	1.66E−01	9.91E−02	2.22E−01	9.47E−08	0.00E+00	1
	CGSA	1.10E−02	6.01E−07	2.16E−02	4.96E−13	0.00E+00	− 1
	MGSA	4.36E−02	3.82E−02	4.86E−02	1.95E−08	00.00E+00	1
	PSO	9.45E+01	2.18E−01	2.25E+02	3.28E−12	0.00E+00	1
	GA	3.82E−02	2.44E−02	3.15E−02	1.44E−03	0.00E+00	1
	Rcr-JADE	1.54E−01	1.12E−01	1.79E−01	6.16E−33	0.00E+00	1
	COBIDE	9.98E+02	1.00E+03	3.62E+02	2.84E+02	0.00E+00	1
9	MCGSA	9.96E−03	3.20E−09	2.28E−02	2.65E−16	NA	NA
	GSA	9.04E−02	4.22E−02	1.45E−01	3.66E−21	0.00E+00	1
	CGSA	9.72E−03	5.14E−10	2.55E−02	1.26E−14	0.00E+00	− 1
	MGSA	2.98E−02	8.58E−05	4.03E−02	2.67E−22	1.08E−49	1
	PSO	9.78E+02	2.36E−01	4.94E+03	4.50E−14	0.00E+00	1
	GA	3.10E−02	2.51E−02	2.83E−02	2.25E−03	0.00E+00	1
	Rcr-JADE	2.91E−01	1.65E−01	4.14E−01	2.33E−22	0.00E+00	1
	COBIDE	1.04E+03	1.02E+03	4.18E+02	4.69E+02	0.00E+00	1
10	MCGSA	4.54E−03	3.57E−11	2.28E−02	6.02E−14	NA	NA
	GSA	2.72E−01	6.72E−02	4.78E−01	3.63E−23	0.00E+00	1
	CGSA	9.93E−03	1.70E−09	2.23E−02	7.69E−14	0.00E+00	1
	MGSA	3.12E−02	7.78E−04	4.11E−02	1.19E−21	0.00E+00	1
	PSO	3.87E+03	2.50E−01	6.83E+03	1.07E−08	0.00E+00	1
	GA	3.56E−02	3.27E−02	2.67E−02	2.43E−03	0.00E+00	1
	Rcr-JADE	5.05E−01	2.92E−01	5.66E−01	4.30E−03	0.00E+00	1
	COBIDE	1.27E+03	1.13E+03	5.33E+02	4.40E+02	0.00E+00	1

Table 3 Statistical results of MSE in a 3-bit parity (XOR) problem (II)

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE	*p* value	h
11	MCGSA	2.68E−03	2.86E−10	1.16E−02	2.11E−14	NA	NA
	GSA	1.33E−01	1.63E−02	2.97E−01	1.13E−22	0.00E+00	1
	CGSA	2.08E−03	9.41E−11	1.14E−02	1.01E−13	0.00E+00	− 1
	MGSA	3.92E−02	8.71E−06	1.57E−01	5.51E−23	2.40E−36	1
	PSO	3.44E+03	1.88E−01	1.32E+04	3.92E−17	0.00E+00	1
	GA	3.23E−02	2.84E−02	2.66E−02	8.28E−04	0.00E+00	1
	Rcr-JADE	5.19E−01	3.20E−01	5.27E−01	1.21E−18	0.00E+00	1
	COBIDE	1.43E+03	1.35E+03	6.03E+02	4.75E+02	0.00E+00	1
13	MCGSA	2.08E−10	8.26E−12	6.14E−10	1.19E−13	NA	NA
	GSA	1.29E−01	3.01E−04	3.32E−01	4.37E−23	0.00E+00	1
	CGSA	1.06E−04	2.05E−11	5.80E−04	1.15E−14	0.00E+00	1
	MGSA	3.18E−03	9.26E−21	1.63E−02	1.23E−23	2.67E−21	1
	PSO	3.57E+03	1.22E+00	6.74E+03	1.31E−10	0.00E+00	1
	GA	4.63E−02	2.59E−02	5.00E−02	6.24E−03	0.00E+00	1
	Rcr-JADE	1.80E+00	1.64E+00	1.29E+00	2.72E−01	0.00E+00	1
	COBIDE	1.54E+03	1.65E+03	5.38E+02	6.61E+02	0.00E+00	1
15	MCGSA	1.93E−09	2.41E−11	7.56E−09	9.44E−14	NA	NA
	GSA	5.08E−02	6.00E−06	1.01E−01	1.84E−22	0.00E+00	1
	CGSA	1.53E−10	1.96E−11	4.71E−10	1.87E−13	0.00E+00	− 1
	MGSA	2.03E−03	5.58E−21	1.08E−02	1.75E−23	0.00E+00	1
	PSO	6.13E+03	2.89E+01	1.34E+04	7.10E−17	0.00E+00	1
	GA	3.93E−02	2.52E−02	3.30E−02	4.78E−03	0.00E+00	1
	Rcr-JADE	3.63E+00	3.02E+00	2.40E+00	5.88E−01	0.00E+00	1
	COBIDE	1.92E+03	1.93E+03	6.73E+02	6.47E+02	0.00E+00	1
20	MCGSA	1.32E−10	4.16E−11	2.40E−10	1.78E−12	NA	NA
	GSA	3.54E−02	1.48E−21	9.76E−02	3.39E−23	9.59E−02	0
	CGSA	1.42E−10	5.34E−11	2.26E−10	7.48E−13	0.00E+00	1
	MGSA	3.55E−03	1.91E−21	1.91E−02	2.05E−22	0.00E+00	1
	PSO	1.99E+04	4.67E+03	2.90E+04	3.54E−16	0.00E+00	1
	GA	3.85E−02	2.64E−02	2.75E−02	5.81E−03	0.00E+00	1
	Rcr-JADE	5.67E+00	5.18E+00	3.28E+00	1.07E+00	0.00E+00	1
	COBIDE	2.37E+03	2.28E+03	8.56E+02	7.97E+02	0.00E+00	1
30	MCGSA	4.35E−10	1.33E−10	8.99E−10	8.33E−12	NA	NA
	GSA	2.04E−01	6.04E−21	6.27E−01	5.64E−22	1.19E−33	1
	CGSA	1.71E−10	1.06E−10	1.97E−10	8.20E−12	0.00E+00	− 1
	MGSA	4.36E−01	8.09E−21	2.38E+00	2.86E−22	0.00E+00	1
	PSO	6.38E+04	3.95E+04	6.75E+04	3.89E−10	0.00E+00	1
	GA	6.35E−02	5.00E−02	4.96E−02	8.97E−03	0.00E+00	1
	Rcr-JADE	9.90E+00	9.00E+00	4.46E+00	3.16E+00	0.00E+00	1
	COBIDE	3.64E+03	3.62E+03	1.14E+03	1.67E+03	0.00E+00	1

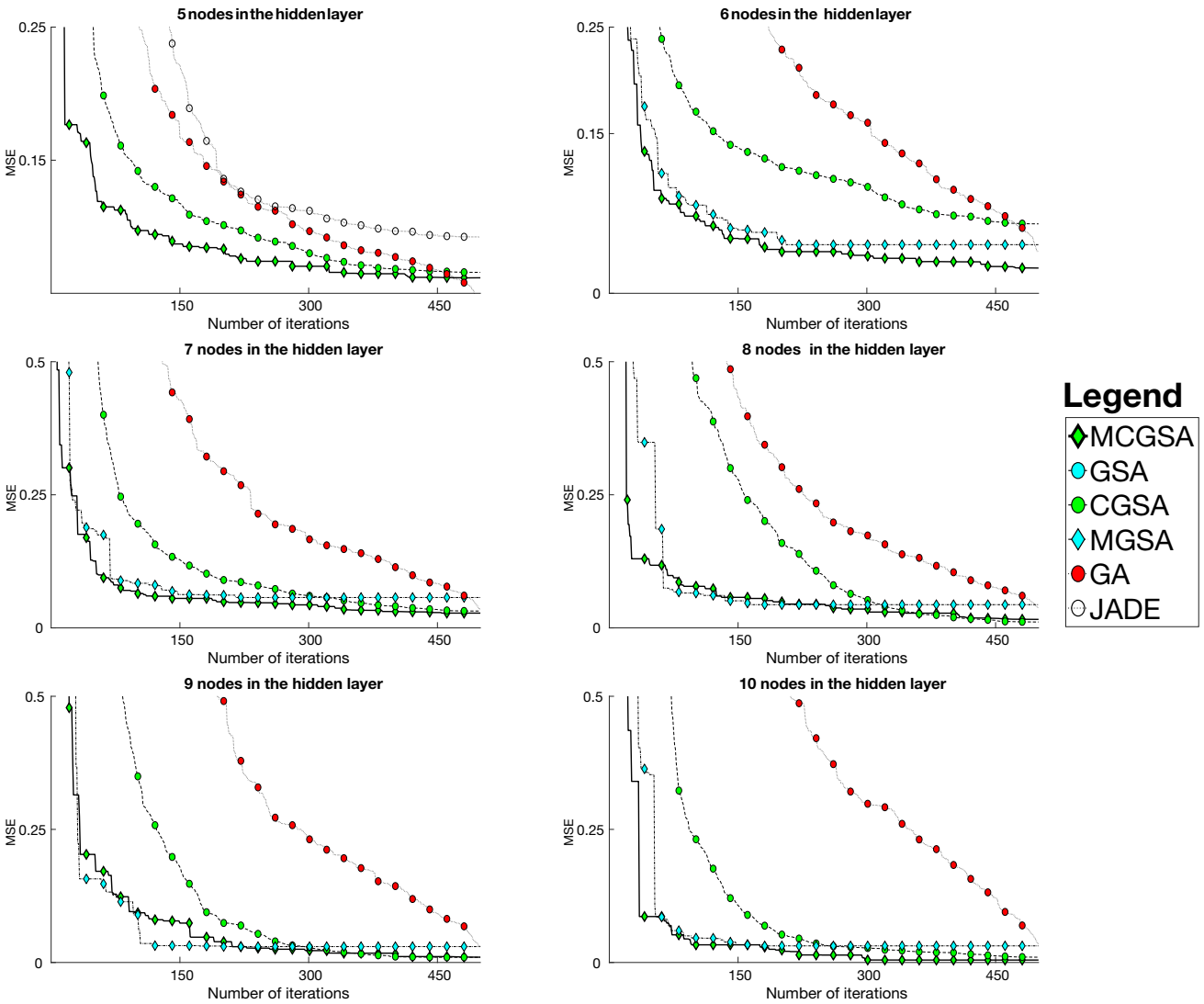

Fig. 8 XOR problem. Convergence curves of the four best algorithms based on averages of MSE for all training samples over 30 independent runs (I)

from the state of the art. The performance of the MCGSA is only outperformed by the GA when $S = 5$ and by the CGSA when $S = \{8, 9, 11, 15, 30\}$. It should also be remarked that the MCGSA maintains its performance despite the increased complexity of the network, which shows the robustness of the algorithm. This does not occur, for example, with Rcr-JADE, which has good performance in the smallest networks, but works worse when bigger networks are considered. Moreover, in order to study not only the final results of the algorithms, but also their convergence to global optima, Figs. 8 and 9 show the average convergence in the 30 runs of the four best algorithms from the eight tested. It is possible to see in these figures that the GSA can achieve good results, but the introduction of qN search in the MGSA speeds up convergence. Moreover, the chaotic component of the MCGSA improves the results obtained by the MGSA, and it is possible to confirm that although the MGSA converges more quickly, it gives a worse solution than the MCGSA, which is capable, thanks to the chaotic component, of avoiding local minima. Furthermore, comparing the MCGSA with the CGSA, the convergence of the MCGSA is quicker than in the case of the CGSA. This is due to the introduction of qN search directions, which improves the exploitation stage of the original CGSA, guaranteeing better convergence. Finally, comparing the results obtained in this study with the results obtained by [45], there is a meaningful improvement, especially in the most complex

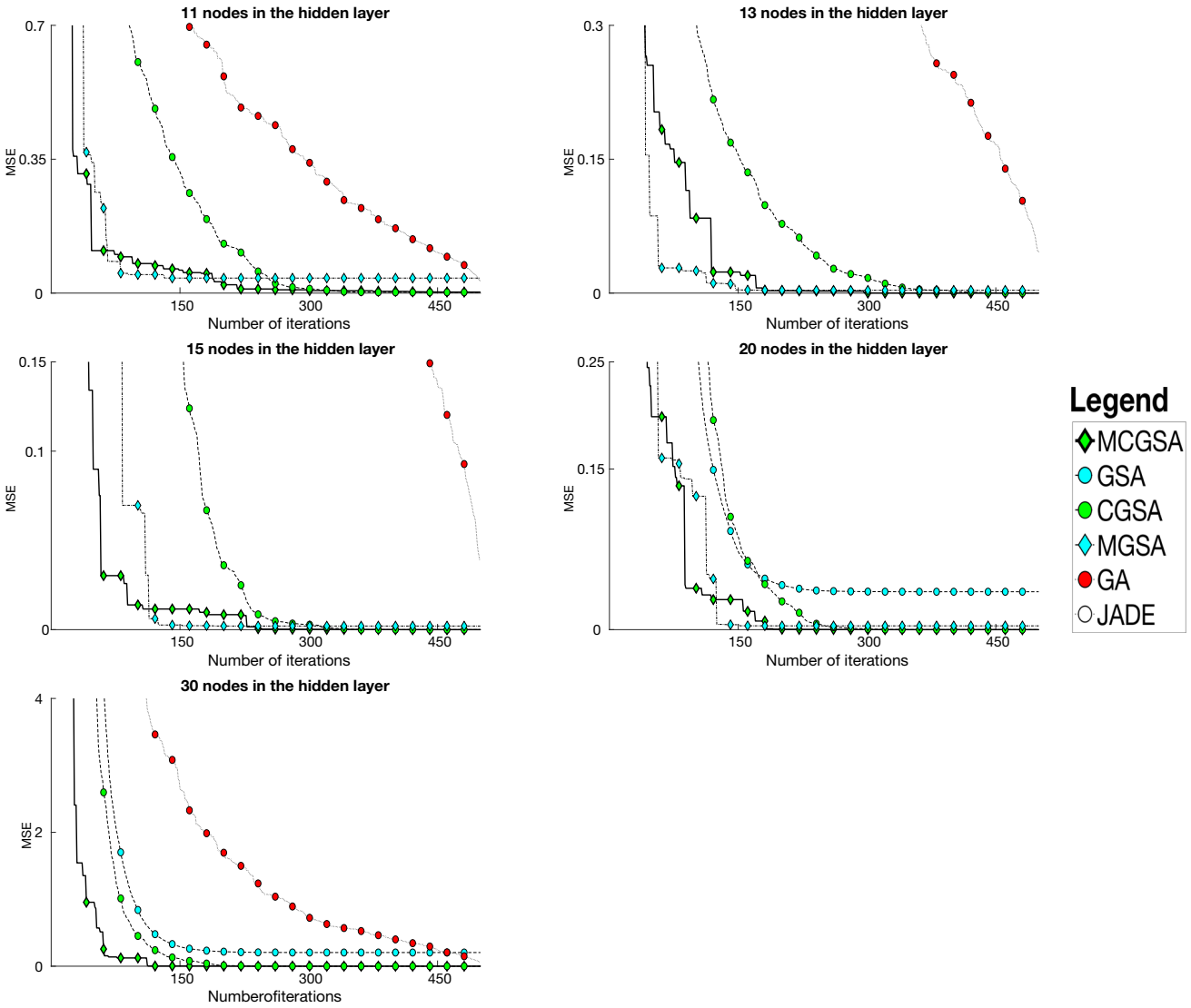

Fig. 9 XOR problem. Convergence curves of the four best algorithms based on averages of MSE for all training samples over 30 independent runs (II)

problems, obtaining better mean and best values with only 60% of functional evaluations in comparison with [45].

5.2 Function approximation problem

One of the most common problems to be addressed when using neural networks is the function approximation problem (see [13, 63, 74]). This is why a problem of this kind has been included in this study. The definition of a function approximation problem is as follows: Given a function $g(x)$ and a set of points from the domain of $g(x)$, called X, the objective is to obtain the image of each point of X which guarantees the most accurate approximation of the function $g(x)$.

In this experiment, the objective function from [45] has been used. Thus, the objective of this experiment is to approximate the function $g(x) = \sin(2x)e^{-x}$ with one dimension. This is done by using a sample of 105 points in the interval $[0, \pi]$, with an increment of 0.03, as a training set. Figure 10 shows the graph of this function with one dimension.

A FNN with the structure 1-S-1 has been used to solve the problem. The input layer will have only one neuron, since the input vector will have only one component (the variable x), and the output layer will have only one neuron, since $g(x)$ is a real function. The hidden layer will have several sizes which will generate different network typologies to be trained. The values of S are:

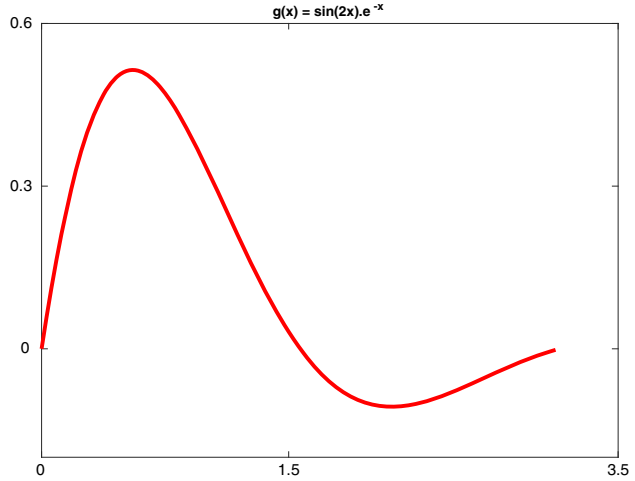

Fig. 10 Target for function approximation problem

$S = \{3, 4, 5, 6, 7\}$. The size of the problems is therefore $n = \{10, 13, 16, 19, 22\}$. The proposed FNN topology for this problem is represented in Fig. 11.

The setup of the experiment is identical to that carried out for the XOR problem. The statistics reported in the results are also the same as for the previous experiments. Again, the Wilcoxon rank-sum test has been used in order to check whether the difference in the performance of two algorithms is statistically significant. The results obtained are shown in Table 4. The MCGSA significantly outperforms the rest of the algorithms considered in the experiment, returning the best results in all problems except the first, where the GA exhibits the best performance. This problem has a total of ten parameters to be adjusted and is the problem of least dimensionality, which could explain why the GA works better in this particular case where the number of parameters is small. However, it is possible to see how, as the number of parameters increases, our proposal continues to perform well in contrast to the GA, whose performance is worsened by increasing the dimensionality of the problem. Another interesting feature of the performance of the MCGSA is the increase in the number of hidden neurons, allowing it to approximate better, giving better values of average MSE error when the number of hidden neurons is larger.

In order to evaluate the behavior of the algorithms over the whole optimization process, Fig. 12 shows the convergence curves of the four best algorithms for each problem. The curves shown represent the mean performance of the four best algorithms in each problem over the 30 runs. As with the XOR problem, the memetic algorithms proposed show the fastest convergence. Although the MGSA converges more quickly than the MCGSA at

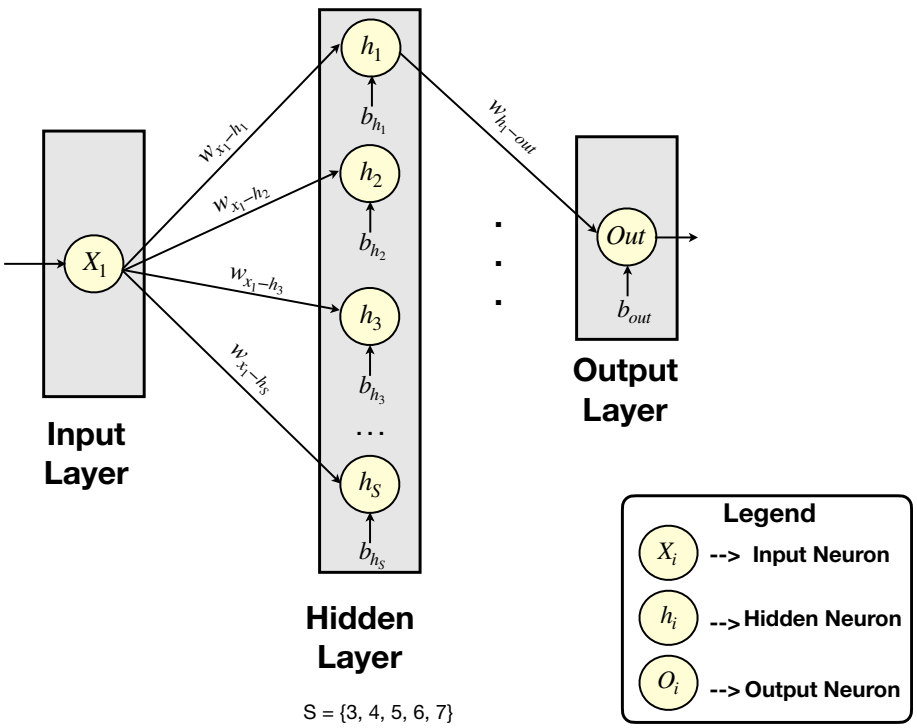

Fig. 11 Graph of the FNN proposed for the function approximation problem

Table 4 Statistical results of MSE in a function approximation problem

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE	*p* value	h
3	MCGSA	9.26E−03	6.72E−03	9.97E−03	4.95E−04	NA	NA
	GSA	2.36E−02	1.54E−02	1.73E−02	5.48E−03	0.00E+00	1
	CGSA	1.38E−02	6.78E−03	1.41E−02	8.57E−04	0.00E+00	1
	MGSA	1.11E−02	7.30E−03	1.17E−02	2.05E−04	1.72E−06	1
	PSO	9.25E−02	4.86E−02	2.53E−01	8.31E−03	0.00E+00	1
	GA	7.61E−03	3.42E−03	1.09E−02	1.33E−03	0.00E+00	− 1
	Rcr-JADE	1.70E−02	1.05E−02	1.54E−02	1.63E−05	0.00E+00	1
	COBIDE	9.40E+00	2.25E+00	1.47E+01	4.87E−02	0.00E+00	1
4	MCGSA	8.32E−03	6.58E−03	7.36E−03	1.14E−04	NA	NA
	GSA	1.87E−02	9.38E−03	1.56E−02	3.11E−03	0.00E+00	1
	CGSA	1.35E−02	7.01E−03	1.40E−02	6.07E−04	0.00E+00	1
	MGSA	8.48E−03	6.72E−03	6.20E−03	6.81E−04	1.05E−01	0
	PSO	4.80E−02	4.86E−02	3.42E−03	2.99E−02	0.00E+00	1
	GA	1.02E−02	6.91E−03	1.03E−02	1.20E−03	0.00E+00	1
	Rcr-JADE	1.86E−02	1.20E−02	1.56E−02	1.32E−03	0.00E+00	1
	COBIDE	2.13E+01	1.01E+01	2.57E+01	5.03E−02	0.00E+00	1
5	MCGSA	6.31E−03	6.32E−03	8.50E−03	2.73E−04	NA	NA
	GSA	2.18E−02	1.33E−02	1.63E−02	2.44E−03	0.00E+00	1
	CGSA	1.02E−02	6.59E−03	1.34E−02	3.77E−04	0.00E+00	1
	MGSA	1.03E−02	7.08E−03	1.07E−02	6.66E−04	0.00E+00	1
	PSO	9.72E−01	4.86E−02	4.31E+00	1.35E−02	0.00E+00	1
	GA	9.05E−03	5.30E−03	1.05E−02	1.31E−03	0.00E+00	1
	Rcr-JADE	1.77E−02	1.22E−02	1.52E−02	6.24E−04	0.00E+00	1
	COBIDE	4.17E+01	3.45E+01	3.12E+01	9.88E−01	0.00E+00	1
6	MCGSA	6.99E−03	6.31E−03	8.70E−03	1.01E−04	NA	NA
	GSA	1.37E−02	8.78E−03	1.34E−02	1.05E−03	0.00E+00	1
	CGSA	9.16E−03	6.56E−03	1.18E−02	8.03E−04	0.00E+00	1
	MGSA	1.12E−02	7.75E−03	1.18E−02	6.38E−04	0.00E+00	1
	PSO	4.49E−02	4.86E−02	1.14E−02	9.30E−03	0.00E+00	1
	GA	7.32E−03	5.40E−03	9.40E−03	1.02E−03	0.00E+00	1
	Rcr-JADE	1.71E−02	1.42E−02	1.26E−02	1.13E−04	0.00E+00	1
	COBIDE	4.81E+01	3.42E+01	4.01E+01	5.85E+00	0.00E+00	1
7	MCGSA	5.24E−03	3.67E−03	7.13E−03	5.59E−04	NA	NA
	GSA	1.90E−02	1.89E−02	1.34E−02	9.49E−04	0.00E+00	1
	CGSA	7.51E−03	6.52E−03	9.49E−03	3.40E−04	0.00E+00	1
	MGSA	7.19E−03	7.15E−03	6.83E−03	5.82E−04	6.54E−19	1
	PSO	1.61E+01	4.86E−02	6.98E+01	1.99E−02	0.00E+00	1
	GA	9.55E−03	4.45E−03	1.26E−02	7.21E−04	0.00E+00	1
	Rcr-JADE	1.75E−02	1.25E−02	1.23E−02	3.73E−04	0.00E+00	1
	COBIDE	7.84E+01	7.63E+01	5.35E+01	6.17E−01	0.00E+00	1

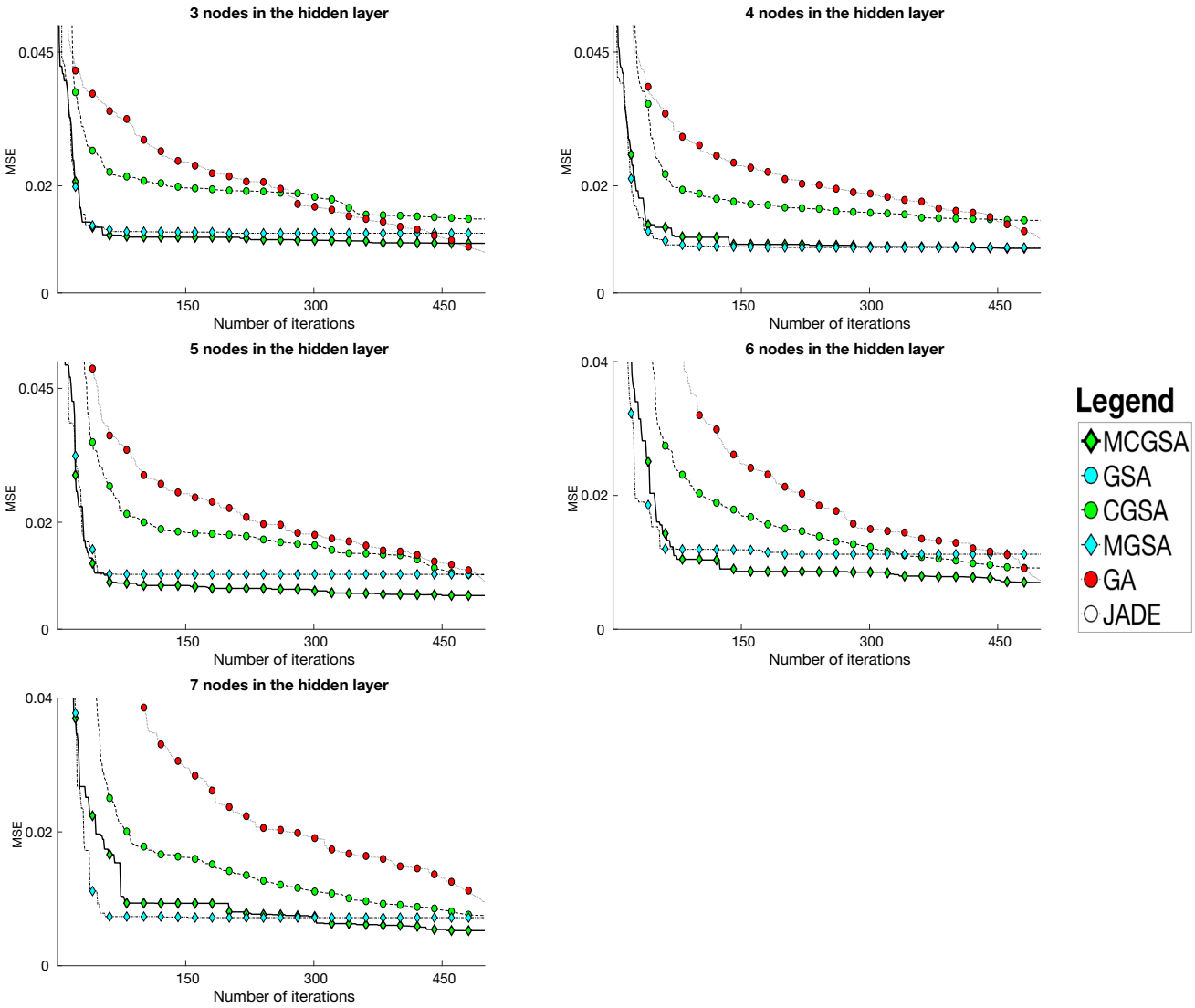

Fig. 12 Approximation function problem. Convergence curves of the four best algorithms based on averages of MSE for all training samples over 30 independent runs

first, the chaotic component of the latter allows it to escape from local optima, ultimately giving the best values. This can be seen, for example, in the curves where four and seven hidden nodes are used. Finally, comparing the results obtained with the results published in [45], it should be remarked that the MCGSA guarantees results of the same order of magnitude, and even improves the results in some cases with only 15,000 function evaluations, in comparison with [45], who obtained their results over 100,000 function evaluations.

5.3 Classification problem

Classification problems are one of the most representative supervised learning problems. This kind of problem can be defined as follows: Given a dataset where each instance is labeled with a class, the aim is to find a model which classifies accurately the instances which belong to each class. The iris or Fisher's iris dataset is one of the most famous such used in classification problems. It has 150 samples of iris flowers. For each one, the following

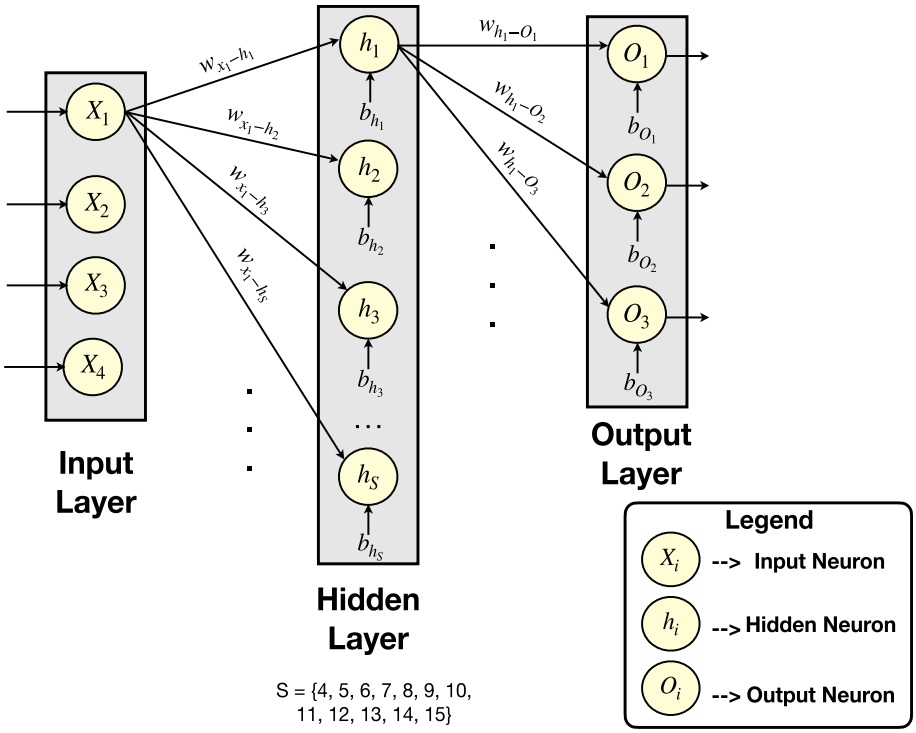

Fig. 13 Graphical representation of the FNN proposed for the iris classification problem

features are known: sepal length, sepal width, petal length and petal width. There are three different classes to which each instance can belong: Iris setosa, Iris versicolor and Iris virginica.

In order to address this problem, a FNN with topology 4-S-3 was designed. Four input neurons are used according to the four features which characterize an instance. The parameter S will again be a variable which will generate the different network topologies to be trained. The values of S are the following: $S = \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. These S values provide the following set of problem dimensions $n = \{27, 35, 43, 51, 59, 67, 75, 83, 91, 99, 107, 115, 123\}$. Finally, the output layer will have three neurons, since there are three possible classes of iris flowers. A graphical representation of the FNN used to solve this problem is shown in Fig. 13.

The configuration of the experiment is the same as for the two previous experiments, returning the same statistics and carrying out the Wilcoxon rank-sum test in order to check whether there are significant differences in the performance of two different algorithms. Left-one cross-validation was also used for training the FNNs, as in [45]. Thus, in each generation, 149 samples of the dataset are chosen for training the FNN, while the unused sample is employed to test the FNN. The statistical results obtained for the training test are shown in Tables 5 and 6.

The results clearly show the superiority of the MCGSA over the rest of the algorithms tested. MCGSA is the algorithm which returns the best mean value in all problems, with the exception of the one with five neurons in the hidden layer, where the MGSA gives the best value. In this problem, where the number of parameters is small, MGSA could speed up the convergence of the algorithm, since it does not introduce chaotic maps to improve the balance between exploration and exploitation, encouraging exploitation through qN search directions.

Another interesting aspect derived from the results is the behavior of the Rcr-JADE and GA. In the XOR problem and function approximation problem, the best algorithms are usually MCGSA, MGSA, CGSA and GA. However, in this problem, Rcr-JADE appears as one of the four best algorithms in the small-size problems (i.e., $S \leq 12$), and

Table 5 Statistical results of MSE in iris classification problem (I)

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE	*p* value	*h*
4	MCGSA	2.25E−01	2.42E−01	1.90E−01	2.46E−02	0.00E+00	NA
	GSA	6.01E−01	4.93E−01	5.25E−01	4.40E−02	0.00E+00	1
	CGSA	4.27E−01	3.70E−01	1.60E−01	9.55E−02	0.00E+00	1
	MGSA	3.80E−01	3.34E−01	1.93E−01	5.78E−02	0.00E+00	1
	PSO	2.46E+00	6.67E−01	5.89E+00	3.33E−01	0.00E+00	1
	GA	6.15E−01	6.56E−01	1.54E−01	3.12E−01	0.00E+00	1
	Rcr-JADE	6.02E−01	6.53E−01	1.14E−01	3.33E−01	0.00E+00	1
	COBIDE	2.43E+03	2.46E+03	8.17E+02	7.46E+02	0.00E+00	1
5	MCGSA	3.46E−01	3.33E−01	1.98E−01	2.25E−02	0.00E+00	NA
	GSA	8.16E−01	4.88E−01	8.59E−01	1.79E−01	0.00E+00	1
	CGSA	3.87E−01	3.46E−01	1.52E−01	9.41E−02	0.00E+00	1
	MGSA	3.16E−01	3.27E−01	3.09E−01	2.01E−02	0.00E+00	− 1
	PSO	9.94E+02	5.19E−01	4.47E+03	3.33E−01	0.00E+00	1
	GA	5.12E−01	5.19E−01	1.67E−01	2.97E−01	0.00E+00	1
	Rcr-JADE	5.93E−01	6.54E−01	1.24E−01	3.19E−01	0.00E+00	1
	COBIDE	3.16E+03	3.05E+03	9.98E+02	1.25E+03	0.00E+00	1
6	MCGSA	2.73E−01	3.29E−01	1.64E−01	1.88E−02	0.00E+00	NA
	GSA	2.12E+00	8.69E−01	3.79E+00	1.72E−01	0.00E+00	1
	CGSA	3.98E−01	3.85E−01	1.46E−01	6.61E−02	0.00E+00	1
	MGSA	4.80E−01	3.29E−01	6.25E−01	3.60E−02	2.68E−01	0
	PSO	7.92E+03	6.67E−01	2.62E+04	3.33E−01	0.00E+00	1
	GA	6.48E−01	6.48E−01	3.55E−01	3.32E−01	0.00E+00	1
	Rcr-JADE	5.85E−01	6.54E−01	1.45E−01	2.79E−01	0.00E+00	1
	COBIDE	3.95E+03	4.07E+03	1.17E+03	1.49E+03	0.00E+00	1
7	MCGSA	2.30E−01	2.36E−01	1.65E−01	2.46E−02	0.00E+00	NA
	GSA	2.22E+00	1.35E+00	2.38E+00	2.04E−01	0.00E+00	1
	CGSA	6.89E−01	4.01E−01	7.68E−01	1.75E−01	0.00E+00	1
	MGSA	3.71E−01	3.02E−01	4.88E−01	2.15E−02	2.35E−02	1
	PSO	6.81E+03	6.15E−01	3.11E+04	1.63E−01	0.00E+00	1
	GA	8.04E−01	6.53E−01	8.48E−01	3.35E−01	0.00E+00	1
	Rcr-JADE	6.07E−01	6.59E−01	2.63E−01	1.10E−01	0.00E+00	1
	COBIDE	4.69E+03	4.72E+03	1.07E+03	2.19E+03	0.00E+00	1
8	MCGSA	2.12E−01	2.16E−01	1.47E−01	2.50E−02	0.00E+00	NA
	GSA	7.41E+00	3.86E+00	9.48E+00	3.68E−01	0.00E+00	1
	CGSA	5.91E−01	5.98E−01	2.62E−01	1.82E−01	0.00E+00	1
	MGSA	7.17E−01	1.65E−01	1.75E+00	3.87E−02	1.74E−41	1
	PSO	1.74E+04	6.67E−01	5.87E+04	3.29E−01	0.00E+00	1
	GA	1.15E+00	6.56E−01	1.37E+00	3.33E−01	0.00E+00	1
	Rcr-JADE	5.43E−01	5.89E−01	3.28E−01	2.72E−02	0.00E+00	1
	COBIDE	5.58E+03	5.78E+03	1.11E+03	2.43E+03	0.00E+00	1
9	MCGSA	1.82E−01	1.61E−01	1.24E−01	2.47E−02	0.00E+00	NA
	GSA	1.50E+01	5.97E+00	2.02E+01	3.08E−01	0.00E+00	1
	CGSA	8.80E−01	5.48E−01	1.18E+00	2.61E−01	0.00E+00	1
	MGSA	4.57E−01	2.09E−01	9.56E−01	2.70E−02	3.29E−16	1
	PSO	8.22E+03	9.21E+01	3.53E+04	3.30E−01	0.00E+00	1
	GA	1.13E+00	7.26E−01	1.01E+00	2.55E−01	0.00E+00	1
	Rcr-JADE	1.03E+00	6.60E−01	1.56E+00	8.81E−02	0.00E+00	1
	COBIDE	6.22E+03	6.36E+03	1.45E+03	3.91E+03	0.00E+00	1

Table 6 Statistical results of MSE in iris classification problem (II)

Hidden nodes (S)	Algorithm	Average MSE	Median MSE	Std dev MSE	Best MSE	p value	h
10	MCGSA	1.97E−01	1.51E−01	1.49E−01	2.70E−02	0.00E+00	NA
	GSA	1.27E+01	8.49E+00	1.19E+01	4.24E−01	0.00E+00	1
	CGSA	1.76E+00	1.08E+00	1.80E+00	8.44E−02	0.00E+00	1
	MGSA	2.87E+00	3.32E−01	8.31E+00	2.46E−02	0.00E+00	1
	PSO	2.96E+03	1.25E+00	6.62E+03	1.07E−01	0.00E+00	1
	GA	1.90E+00	1.36E+00	1.41E+00	3.76E−01	0.00E+00	1
	Rcr-JADE	7.14E−01	3.95E−01	1.04E+00	1.07E−01	0.00E+00	1
	COBIDE	7.38E+03	7.38E+03	1.37E+03	4.38E+03	0.00E+00	1
11	MCGSA	2.69E−01	1.15E−01	4.77E−01	2.76E−02	0.00E+00	NA
	GSA	1.30E+01	1.08E+01	1.19E+01	9.46E−01	0.00E+00	1
	CGSA	2.00E+00	1.31E+00	1.96E+00	2.53E−01	0.00E+00	1
	MGSA	5.08E−01	2.04E−01	1.19E+00	2.55E−02	1.69E−70	1
	PSO	8.06E+03	4.48E+00	3.62E+04	1.94E−01	0.00E+00	1
	GA	3.40E+00	2.53E+00	2.25E+00	1.15E+00	0.00E+00	1
	Rcr-JADE	1.53E+00	4.84E−01	3.34E+00	7.53E−02	0.00E+00	1
	COBIDE	7.90E+03	7.66E+03	1.82E+03	4.40E+03	0.00E+00	1
12	MCGSA	1.88E−01	1.75E−01	1.39E−01	1.74E−02	0.00E+00	NA
	GSA	1.78E+01	1.65E+01	1.19E+01	4.55E−01	0.00E+00	1
	CGSA	3.44E+00	2.45E+00	3.12E+00	4.05E−01	0.00E+00	1
	MGSA	4.84E+00	1.76E−01	1.03E+01	5.30E−02	0.00E+00	1
	PSO	1.92E+04	3.76E+02	5.86E+04	1.73E−01	0.00E+00	1
	GA	2.78E+00	2.26E+00	1.77E+00	4.57E−01	0.00E+00	1
	Rcr-JADE	5.51E+00	7.08E−01	1.14E+01	1.51E−01	0.00E+00	1
	COBIDE	7.89E+03	7.84E+03	1.96E+03	4.50E+03	0.00E+00	1
13	MCGSA	5.46E−01	1.76E−01	1.43E+00	2.58E−02	0.00E+00	NA
	GSA	1.86E+01	1.71E+01	1.37E+01	1.97E+00	0.00E+00	1
	CGSA	4.21E+00	3.62E+00	3.35E+00	3.19E−01	0.00E+00	1
	MGSA	7.63E+00	2.05E−01	1.52E+01	3.04E−02	0.00E+00	1
	PSO	7.20E+03	4.34E+03	7.49E+03	2.16E−01	0.00E+00	1
	GA	3.19E+00	2.95E+00	2.16E+00	6.92E−01	0.00E+00	1
	Rcr-JADE	8.89E+00	8.22E−01	1.97E+01	2.10E−01	0.00E+00	1
	COBIDE	9.17E+03	9.52E+03	2.06E+03	4.43E+03	0.00E+00	1
14	MCGSA	3.50E−01	1.52E−01	9.57E−01	2.67E−02	0.00E+00	NA
	GSA	3.13E+01	2.44E+01	2.55E+01	2.98E+00	0.00E+00	1
	CGSA	7.11E+00	4.29E+00	7.31E+00	8.28E−01	0.00E+00	1
	MGSA	1.12E+00	2.28E−01	3.45E+00	4.44E−02	0.00E+00	1
	PSO	1.02E+04	3.32E+03	1.63E+04	3.27E−01	0.00E+00	1
	GA	4.66E+00	4.80E+00	2.31E+00	3.45E−01	0.00E+00	1
	Rcr-JADE	1.87E+01	3.93E+00	2.70E+01	2.06E−01	0.00E+00	1
	COBIDE	1.07E+04	1.05E+04	2.38E+03	6.62E+03	0.00E+00	1
15	MCGSA	1.68E−01	1.72E−01	7.47E−02	3.94E−02	0.00E+00	NA
	GSA	4.29E+01	3.76E+01	3.02E+01	5.25E+00	0.00E+00	1
	CGSA	4.39E+00	3.33E+00	3.90E+00	3.31E−01	0.00E+00	1
	MGSA	5.13E+00	1.68E−01	1.16E+01	1.78E−02	8.33E−63	1
	PSO	6.08E+04	1.09E+04	1.54E+05	1.96E+01	0.00E+00	1
	GA	5.23E+00	4.65E+00	3.10E+00	8.06E−01	0.00E+00	1
	Rcr-JADE	2.56E+01	5.73E+00	4.24E+01	1.58E−01	0.00E+00	1
	COBIDE	1.12E+04	1.10E+04	1.77E+03	8.43E+03	0.00E+00	1

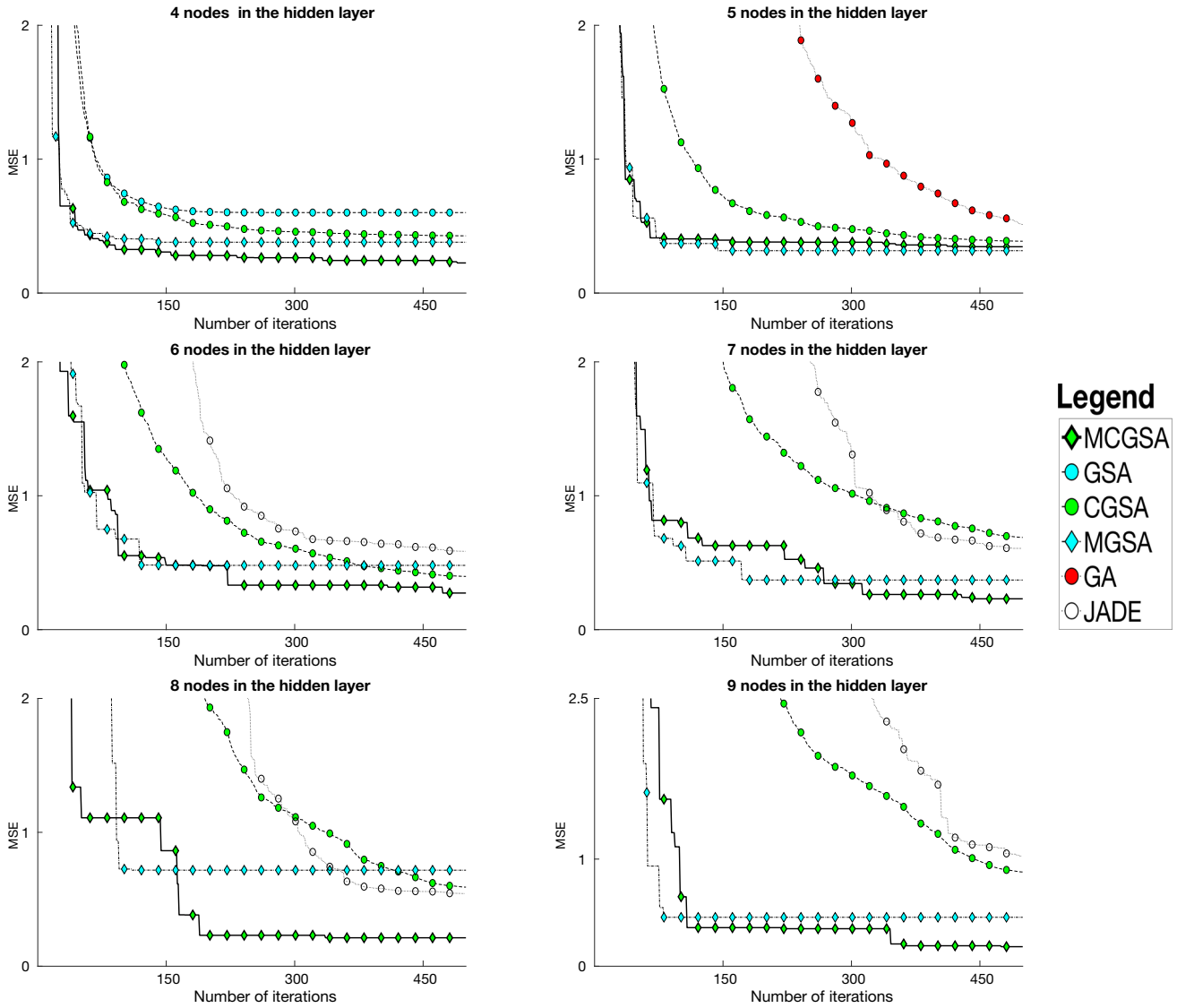

Fig. 14 Iris problem. Convergence curves of the four best algorithms based on averages of MSE for all training samples over 30 independent runs (I)

then, GA emerges as one of the best algorithm, replacing the other. This implies that Rcr-JADE is a good choice in small instances, while GA is more appropriate when considering larger sizes. As in the previous experiments, the convergence curves are shown in Figs. 14 and 15, with the purpose of analyzing the performance of the algorithms over the whole optimization process. It is possible to check that the memetic proposals show the best convergences curves. Again, and as in the previous experiments, although MGSA converges more quickly than MCGSA, the latter is capable of avoiding local minima due to the chaotic maps introduced by the G constant, showing the best performance of all the algorithms tested.

6 Conclusions and further work

This paper applies two recently proposed MAs based on qN search directions to the FNN neural network training problem in order to study whether the memetic approach can improve the performance of metaheuristic algorithms when training such networks.

To do this, MGSA and MCGSA were applied to three classical benchmark problems, along with classical GSA and CGSA and other metaheuristics from the state of the art, such as PSO, GA, Rcr-JADE and COBIDE. The main conclusions arising from the computational experiments undertaken in this study are:

– The results obtained by MCGSA applied to the FNN neural network training problem show this is the

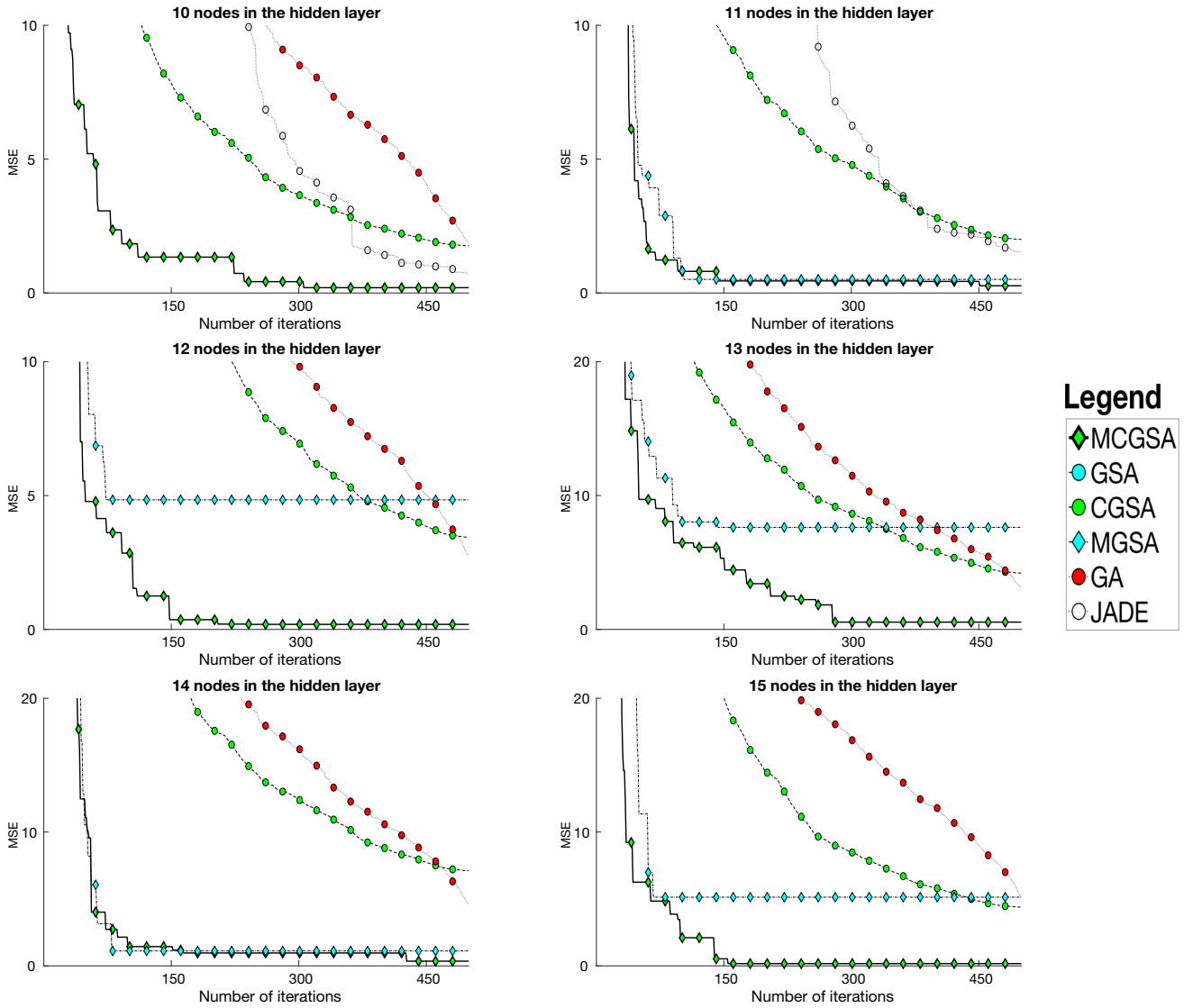

Fig. 15 Iris problem. Convergence curves of the four best algorithms based on averages of MSE for all training samples over 30 independent runs (II)

algorithm with the best performance among all the tested algorithms in the XOR problem, the function adjustment problem and the classification problem. The improvement achieved by MCGSA can be seen both in the speed of convergence and in the quality of the solution obtained.

– The CGSA is capable of avoiding local optima, due to the sinusoidal chaotic map added to it. This property is inherited by MCGSA, while the introduction of the qN algorithm in CGSA, which has a superlinear convergence rate, improves the convergence speed of the baseline algorithm. The improvement is especially evident in the function approximation problem and in the classification problem. It is also possible to see that the *curse of dimensionality* affects the MCGSA less than the CGSA, obtaining better solutions by an order of magnitude in the MSE, for the classification problems of greater dimensionality.
– The problem of metaheuristic algorithms in addressing the *curse of dimensionality* problem has been noted. MCGSA is the algorithm that best addresses this problem.

This work has focused on the use of qN as a local search method. However, the framework of MA allows the inclusion of specialized algorithm for the FNN neural network training problem, such as BP. These alternatives should be explored.

Moreover, more effort must be made with regard to data quality, since the effectiveness of a FNN depends to a great extent on the data used to train it. Thus, problems like imbalanced data, insufficient or overabundant data and

high-dimensional data must be addressed in order to improve the performance of FNNs (see [43]).

Currently, in the era of big data, stream data (see [76]) are a kind of high-dimensional data that are very common in many areas, such as natural language processing, speech processing and social network data. Although the processing of these data is analyzed through deep learning architectures, FNNs trained with MAs can be tested in order to study performance. Also, memetic FNNs could be useful in managing and reducing the data stream (see [30]).

Furthermore, the use of evolutionary algorithms for optimizing the topology of deep neural networks is currently a hot topic in the literature. Future work should look at specializing memetic algorithms for optimizing the topology of deep and convolutional neural networks.

Finally, regarding the data stream provided by internet of things (IoT) devices, such as human activity data, social activity data or smartphone data, these need to be processed using inexpensive complex models (see [59]). In this context, tools such as memetic FNNs can be a suitable choice for analyzing these data.

Acknowledgements This work was supported by *Spanish Ministry of Economy, Industry and Competitiveness*-FEDER EU grants with number TRA2016-76914-C3-2-P. The authors would like to thank professor A. P. Piotrowski, for sharing the source codes of Rcr-JADE and COBIDE algorithms.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abd-Elazim SM, Ali ES (2013) A hybrid particle swarm optimization and bacterial foraging for optimal power system stabilizers design. Int J Electr Power Energy Syst 46(1):334–341
2. Abd Elazim SM, Ali ES (2016) Optimal SSSC design for damping power systems oscillations via gravitational search algorithm. Int J Electr Power Energy Syst 82:161–168
3. Aldhafferi N, Owolabi TO, Akande KO, Olatunji SO, Alqahtani A (2018) Development of hybrid computational intelligence model for estimating relative cooling power of manganite-based materials for magnetic refrigeration enhancement. J Eng Appl Sci 13(6):1575–1583
4. Arora, S., Cohen, N., Golowich, N., Hu, W.: A convergence analysis of gradient descent for deep linear neural networks. CoRR **abs/1810.0** (2018)
5. Azali S, Sheikhan M (2016) Intelligent control of photovoltaic system using BPSO-GSA-optimized neural network and fuzzy-based PID for maximum power point tracking. Appl Intell 44(1):88–110
6. Ballings M, Van Den Poel D, Hespeels N, Gryp R (2015) Evaluating multiple classifiers for stock price direction prediction. Exp Syst Appl 42(20):7046–7056
7. Bohte S, Kok J, La Poutré H (2002) Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing 48(1–4):17–37
8. Chady T, Lopato P (2007) Flaws identification using an approximation function and artificial neural networks. IEEE Trans Magn 43(4):1769–1772
9. Cheng CT, Lin JY, Sun YG, Chau K (2005) Long-term prediction of discharges in Manwan hydropower using adaptive-network-based fuzzy inference systems models. In: Wang L, Chen K, Ong YS (eds) Advances in natural computation. Springer, Berlin, pp 1152–1161
10. Dennis, J.E., Moré, J.J.: Quasi-Newton Methods, Motivation and Theory (1974)
11. Derrac J, García S, Hui S, Suganthan P, Herrera F (2014) Analyzing convergence performance of evolutionary algorithms: a statistical approach. Inf Sci 289(1):41–58
12. Dhahri H, Alimi AM, Abraham A (2013) Hierarchical particle swarm optimization for the design of beta basis function neural network. In: Abraham A, Thampi SM (eds) Intelligent informatics. Springer, Berlin, pp 193–205
13. El-Melegy M (2013) Random sampler m-estimator algorithm with sequential probability ratio test for robust function approximation via feed-forward neural networks. IEEE Trans Neural Netw Learn Syst 24(7):1074–1085
14. Espinosa-Aranda J, García-Ródenas R, Ramírez-Flores M, López-García M, Angulo E (2015) High-speed railway scheduling based on user preferences. Eur J Oper Res 246(3):772–786
15. Fahlman SE, Lebiere C (1990) Advances in neural information processing systems 2. chap. The Cascad. Morgan Kaufmann Publishers Inc, San Francisco, pp 524–532
16. Fernández-Delgado M, Cernadas E, Barro S, Amorim D (2014) Do we need hundreds of classifiers to solve real world classification problems? J Mach Learn Res 15:3133–3181
17. Fotovatikhah F, Herrera M, Shamshirband S, Chau KW, Ardabili SF, Piran MJ (2018) Survey of computational intelligence as basis to big flood management: challenges, research directions and future work. Eng Appl Comput Fluid Mech 12(1):411–437
18. García S, Fernández A, Luengo J, Herrera F (2010) Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf Sci 180:2044–2064
19. García-Ródenas R, Linares L, López-Gómez J (2017) On the performance of classic and deep neural models in image recognition, vol 10585. LNCS, Berlin
20. García-Ródenas R, Linares LJ, López-Gómez JA (2019) A memetic chaotic gravitational search algorithm for unconstrained global optimization problems. Appl Soft Comput
21. Gardner WA (1984) Learning characteristics of stochastic-gradient-descent algorithms: a general study, analysis, and critique. Signal Process 6(2):113–133
22. Ghosh-Dastidar S, Adeli H (2009) A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. Neural Netw 22(10):1419–1431

23. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: In Proceedings of the international conference on artificial intelligence and statistics (AISTATS'10). Society for Artificial Intelligence and Statistics (2010)
24. Gong W, Cai Z, Wang Y (2014) Repairing the crossover rate in adaptive differential evolution. Appl Soft Comput J 15:149–168
25. Gong W, Fialho A, Cai Z, Li H (2011) Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. Inf Sci 181(24):5364–5386
26. Gori M, Tesi A (1992) On the problem of local minima in backpropagation. IEEE Trans Pattern Anal Mach Intell 14(1):76–86
27. Guo SM, Tsai JH, Yang CC, Hsu PH (2015) A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover and successful-parent-selecting framework on CEC 2015 benchmark set. In: 2015 IEEE congress on evolutionary computation, CEC 2015—Proceedings, pp 1003–1010
28. Hagenauer J, Helbich M (2017) A comparative study of machine learning classifiers for modeling travel mode choice. Expert Syst Appl 78:273–282
29. Han F, Jiang J, Ling QH, Su BY (2018) A survey on metaheuristic optimization for random single-hidden layer feedforward neural network. Neurocomputing 335:261–273
30. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
31. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
32. Hush DR, Horne BG (1993) Progress in supervised neural networks. IEEE Signal Process Mag 10(1):8–39
33. Irie M (1988) Capabilities of three-layered perceptrons. In: IEEE 1988 international conference on neural networks, pp 641–648
34. Juang CF (2004) A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans Syst Man Cybern B (Cybern) 34(2):997–1006
35. Karaboga D, Akay B (2009) A comparative study of artificial bee colony algorithm. Appl Math Comput 214(1):108–132
36. Khadanga RK, Satapathy JK (2015) A new hybrid GA-GSA algorithm for tuning damping controller parameters for a unified power flow controller. Int J Electr Power Energy Syst 73:1060–1069
37. Larochelle H, Bengio Y, Louradour J, Lamblin P (2009) Exploring strategies for training deep neural networks. J Mach Learn Res 10:1–40
38. LeCun Y, Bengio Y, Hinton G (2015) Nature 521
39. Liu DC, Nocedal J (1989) On the limited memory BFGS method for large scale optimization. Math Program 45(1):503–528
40. García Luz López M, García-Ródenas R, González Gómez A (2015) K-means algorithms for functional data. Neurocomputing 151(P1):231–245
41. Malakooti B, Zhou Y (1998) Approximating polynomial functions by feedforward artificial neural networks: capacity analysis and design. Appl Math Comput 90(1):27–51
42. Mat Isa NA, Mamat WMFW (2011) Clustered-hybrid multilayer perceptron network for pattern recognition application. Appl Soft Comput 11(1):1457–1466
43. Mazurowski MA, Habas PA, Zurada JM, Lo JY, Baker JA, Tourassi GD (2008) Training neural network classifiers for medical decision making: the effects of imbalanced datasets on classification performance. Neural Netw 21(2–3):427–436
44. Mirjalili S, Gandomi A (2017) Chaotic gravitational constants for the gravitational search algorithm. Appl Soft Comput J 53:407–419
45. Mirjalili S, Mohd Hashim S, Moradian Sardroudi H (2012) Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Appl Math Comput 218(22):11125–11137
46. Moazenzadeh R, Mohammadi B, Shamshirband S, Chau KW (2018) Coupling a firefly algorithm with support vector regression to predict evaporation in northern Iran. Eng Appl Comput Fluid Mech 12(1):584–597
47. Moscato P (1999) New ideas in optimization, chap. Memetic Al. McGraw-Hill Ltd., Maidenhead, pp 219–234
48. Najafi B, Ardabili SF, Shamshirband S, Chau KW, Rabczuk T (2018) Application of ANNs, ANFIS and RSM to estimating and optimizing the parameters that affect the yield and cost of biodiesel production. Eng Appl Comput Fluid Mech 12(1):611–624
49. Najafzadeh M, Azamathulla HM (2015) Neuro-fuzzy GMDH to predict the scour pile groups due to waves. J Comput Civil Eng 29(5)
50. Neri F, Cotta C (2012) Memetic algorithms and memetic computing optimization: a literature review. Swarm Evolut Comput 2:1–14
51. Nocedal J (1980) Updating quasi-Newton matrices with limited storage. Math Comput 35(151):773–782
52. Ojha VK, Abraham A, Snášel V (2017) Metaheuristic design of feedforward neural networks: a review of two decades of research. Eng Appl Artif Intell 60:97–116
53. Ojha VK, Abraham A, Snášel V (2014) Simultaneous optimization of neural network weights and active nodes using metaheuristics. In: 2014 14th international conference on hybrid intelligent systems, pp 248–253
54. Omrani H (2015) Predicting travel mode of individuals by machine learning. In: Transportation research procedia, vol 10. Elsevier, Amsterdam, pp 840–849
55. Owolabi TO, Gondal MA (2017) A hybrid intelligent scheme for estimating band gap of doped titanium dioxide semiconductor using crystal lattice distortion. Comput Mater Sci 137:249–256
56. Owolabi TO, Gondal MA (2018) Development of hybrid extreme learning machine based chemo-metrics for precise quantitative analysis of LIBS spectra using internal reference pre-processing method. Anal Chim Acta 1030:33–41
57. Piotrowski A, Napiorkowski M, Napiorkowski J, Rowinski P (2017) Swarm intelligence and evolutionary algorithms: performance versus speed. Inf Sci 384:34–85
58. Pošík P, Huyer W, Pál L (2012) A comparison of global search algorithms for continuous black-box optimization. In: Evolutionary computation, 20
59. Prisecaru P (2016) Challenges of the fourth industrial revolution. Knowl Horizons Econ 8(1):57–62
60. Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179(13):2232–2248
61. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In: IEEE international conference on neural networks, pp 586–591 (1993)
62. Sheikhpour S, Sabouri M, Zahiri SH (2013) A hybrid gravitational search algorithm genetic algorithm for neural network training. In: 2013 21st Iranian conference on electrical engineering, ICEE 2013
63. Silva P, Fernandes E, Neto A (2002) A feed forward neural network with resolution properties for function approximation and modeling. In: Proceedings-Brazilian symposium on neural networks, SBRN, vol 2002-Jan, pp 55–60
64. Tanabe R, Fukunaga A (2013) Success-history based parameter adaptation for differential evolution. In: 2013 IEEE congress on evolutionary computation, CEC 2013, pp 71–78
65. Tsai JT, Chou JH, Liu TK (2006) Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm. IEEE Trans Neural Netw 17(1):69–80
66. Marquardt DW (1963) An algorithm for least square estimation of non-linear parameters. SIAM J Appl Math 11:431–441

67. Wang S, Zhang Y, Ji G, Yang J, Wu J, Wei L (2015) Fruit classification by wavelet-entropy and feedforward neural network trained by fitness-scaled chaotic abc and biogeography-based optimization. Entropy 17(8):5711–5728
68. Wc Wang, Kw Chau, Qiu L, Yb Chen (2015) Improving forecasting accuracy of medium and long-term runoff using artificial neural network based on EEMD decomposition. Environ Res 139:46–54
69. Wang Y, Li HX, Huang T, Li L (2014) Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. Appl Soft Comput J 18:232–247
70. Yaghini M, Khoshraftar MM, Fallahi M (2013) A hybrid algorithm for artificial neural network training. Eng Appl Artif Intell 26(1):293–301
71. Yao X (1993) A review of evolutionary artificial neural networks. Int J Intell Syst 8(4):539–567
72. Yaseen ZM, Sulaiman SO, Deo RC, Chau KW (2019) An enhanced extreme learning machine model for river flow forecasting: state-of-the-art, practical applications in water resource engineering area and future research direction. J Hydrol 569:387–408
73. Yin F, Mao H, Hua L (2011) A hybrid of back propagation neural network and genetic algorithm for optimization of injection molding process parameters. Mater Design 32(6):3457–3464
74. Zainuddin Z, Ong P (2008) Function approximation using artificial neural networks. WSEAS Trans Math 7(6):333–338
75. Zhang JR, Zhang J, Lok TM, Lyu MR (2007) A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. Appl Math Comput 185(2):1026–1037
76. Zikopoulos P, Eaton C (2011) Understanding big data: analytics for enterprise class hadoop and streaming data, 1st edn. McGraw-Hill, New York