Neurocomputing 129 (2014) 428–436

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Learning of a single-hidden layer feedforward neural network using an optimized extreme learning machine

Tiago Matias [a,b,*], Francisco Souza [a,b], Rui Araújo [a,b], Carlos Henggeler Antunes [b,c]

[a] Institute of Systems and Robotics (ISR-UC), University of Coimbra, Pólo II, PT-3030-290 Coimbra, Portugal
[b] Department of Electrical and Computer Engineering (DEEC-UC), University of Coimbra, Pólo II, PT-3030-290 Coimbra, Portugal
[c] Institute for Systems Engineering and Computers (INESC), R. Antero de Quental, 199, PT-3000-033 Coimbra, Portugal

ARTICLE INFO

Article history:
Received 18 January 2013
Received in revised form
1 August 2013
Accepted 2 September 2013
Communicated by A. Abraham
Available online 23 October 2013

Keywords:
Optimized extreme learning machine
Single-hidden layer feedforward neural networks
Genetic algorithms
Simulated annealing
Differential evolution

ABSTRACT

This paper proposes a learning framework for single-hidden layer feedforward neural networks (SLFN) called optimized extreme learning machine (O-ELM). In O-ELM, the structure and the parameters of the SLFN are determined using an optimization method. The output weights, like in the batch ELM, are obtained by a least squares algorithm, but using Tikhonov's regularization in order to improve the SLFN performance in the presence of noisy data. The optimization method is used to the set of input variables, the hidden-layer configuration and bias, the input weights and Tikhonov's regularization factor. The proposed framework has been tested with three optimization methods (genetic algorithms, simulated annealing, and differential evolution) over 16 benchmark problems available in public repositories.

1. Introduction

Multilayer feedforward neural networks (FFNN) have been used in the identification of unknown linear or non-linear systems (see, e.g. [1,2]). Their appeal is based on their universal approximation properties [3,4]. However, in industrial applications, linear models are often preferred due to faster training in comparison with multilayer FFNN trained with gradient-descent algorithms [5]. In order to overcome the slow construction of FFNN models, a new method called extreme learning machine (ELM) is proposed in [6]. This method is a new batch learning algorithm for single-hidden layer FFNN (SLFN) where the input weights (weights of connections between the input variables and neurons in the hidden-layer) and the bias of neurons in the hidden-layer are randomly assigned. The output weights (weights of connections between the neurons in the hidden-layer and the output neuron) are obtained using the Moore–Penrose (MP) generalized inverse, considering that the activation function of the output neuron is linear.

Since in ELM the output weights are computed based on the random input weights and bias of the hidden nodes, there may exist a set of non-optimal or unnecessary input weights and bias of the hidden nodes. Furthermore, the ELM tends to require more hidden neurons than conventional tuning-based learning algorithms (based on error backpropagation or other learning methods where the output weights are not obtained by the least squares method) in some applications, which can negatively affect SLFN performance in unknown testing data [6]. The use of the least squares method without regularization in noisy data also makes the model displaying a poor generalization capability [7]. Fitting problems may also be encountered in the presence of irrelevant or correlated input variables [5].

Optimization methods have been used jointly with analytical methods for network training. In [8] a new method to choose the most appropriate FFNN topology, type of activation functions and parameters of the training algorithm using a genetic algorithm (GA) is proposed. Each chromosome is composed of the specification of the minimization algorithm used in the back-propagation (BP) method, the network architecture, the activation function of the neurons of the hidden layer, and the activation function of the neurons of the output layer using binary encoding. In [9] a new nonlinear system identification scheme is proposed, where differential evolution (DE) is used to optimize the initial weights used by a Levenberg–Marquardt (LM) algorithm in the learning of a FFNN. In [10] a similar method is proposed using a simulated annealing (SA)

* Corresponding author at: Department of Electrical and Computer Engineering (DEEC-UC), University of Coimbra, Pólo II, PT-3030-290 Coimbra, Portugal.
Tel.: +351 96 1569024.
E-mail addresses: tmatias@isr.uc.pt (T. Matias), fasouza@isr.uc.pt (F. Souza), rui@isr.uc.pt (R. Araújo), ch@deec.uc.pt (C.H. Antunes).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.09.016

approach. In these algorithms, the evaluation of each individual or state is made by training the FFNN with a BP method, which is computationally expensive.

In [11] an improved GA is used to optimize the structure (connections layout) and the parameters (connection weights and biases) of a SLFN with switches. The switches are unit step functions that make possible the removal of each connection. Using a real encoding scheme, and new crossover and mutation techniques, this improved GA obtains better results in comparison with traditional GAs. The structure and the parameters of the same kind of SLFN with switches are also tuned in [12], in this case using a hybrid Taguchi GA. This approach is similar to a traditional GA but a Taguchi method [13] is used for the crossover process. The use of this method implies the construction of a $(n+1)\times n$ two-level orthogonal matrix, where n is the number of variables for the optimization process. However, the construction of this orthogonal matrix is not simple. There are some standard orthogonal matrices but they can be only used when n is small. In large networks, n is large and therefore this method is not a good practical approach. In these methodologies, the weights between the hidden-layer and the output layer are optimized by the GA. Using the ELM approach, the output weights could be calculated using the Moore–Penrose generalized inverse (considering an output neuron with linear activation function) and a good solution could be quickly obtained, reducing the convergence time of the GA. Furthermore, as the number of variables of the optimization process is lower, the search space to be explored by the GA narrows. This approach was used in [14] where a GA is used to tune the (selective existence of) connections and parameters between the input layer and the hidden layer, and a least squares algorithm is applied to tune the parameters between the hidden layer and the output layer. However, in this type of approach it is difficult to deal with the tendency to require more hidden nodes than conventional tuning-based algorithms, as well as the problem caused by the presence of irrelevant variables is difficult to solve. These problems occur also in the methods proposed in [15] and [16]. In [15] a learning method called evolutionary ELM (E-ELM) is proposed, where the weights between the input layer and the hidden layer and the bias of the hidden layer neurons are optimized by a DE method and the output weights are calculated using the Moore–Penrose generalized inverse like in ELM. In [16] a similar method called self-adaptive E-ELM (SaE-ELM) is proposed; however, in this methodology the generation strategies and control parameters of the DE method are self-adapted by the optimization method.

In this paper, a novel learning framework for SLFNs called optimized extreme learning machine (O-ELM) is proposed. This framework uses the same concept of the ELM where the output weights are obtained using least squares, however, with the difference that Tikhonov's regularization [17] is used in order to obtain a robust least squares solution. The problem of reduction of the ELM performance in the presence of irrelevant variables is well known, as well as its propensity for requiring more hidden nodes than conventional tuning-based learning algorithms. To solve these problems, the proposed framework uses an optimization method to select the set of input variables and the configuration of the hidden-layer. Furthermore, in order to optimize the fitting performance, the optimization method also selects the weights of connections between the input layer and the hidden-layer, the bias of neurons of the hidden-layer, and the regularization factor. Using this framework, no trial-and-error experiments are needed to search for the best SLFN structure. In this paper, three optimization methods (GA, SA, and DE) are tested in the proposed framework.

selection of the optimal number of neurons in this layer and the activation function of each neuron, trying to overcome the propensity of ELM for requiring more hidden nodes than conventional tuning-based learning algorithms. In this paper, three optimization methods (GA, SA, and DE) are tested in the proposed framework.

The paper is organized as follows. The SLFN architecture is overviewed in Section 2. Section 3 gives a brief review of the batch ELM. The proposed learning framework is presented in Section 4. Section 5 gives a brief review of the optimization methods tested in the O-ELM. Section 6 presents experimental results. Finally, concluding remarks are drawn in Section 7.

2. Adjustable single hidden-layer feedforward network architecture

The neural network considered in this paper is a SLFN with adjustable architecture as shown in Fig. 1, which can be mathematically represented by

$$y=g\left(b_O+\sum_{j=1}^{h} w_{jO}v_j\right), \tag{1}$$

$$v_j=f_j\left(b_j+\sum_{i=1}^{n} w_{ij}s_ix_i\right). \tag{2}$$

n and h are the number of input variables and the number of the hidden layer neurons, respectively; v_j is the output of the hidden-layer neuron j; x_i, $i=1,\ldots,n$, are the input variables; w_{ij} is the weight of the connection between the input variable i and the neuron j of the hidden layer; w_{jO} is the weight of the connection between neuron j of the hidden layer and the output neuron; b_j is the bias of the hidden layer neuron j, $j=1,\ldots,h$, and b_O is the bias of the output neuron; $f_j(\cdot)$ and $g(\cdot)$ represent the activation function of the neuron j of the hidden layer and the activation function of the output neuron, respectively. s_i is a binary variable used in the selection of the input variables during the design of the SLFN.

Using the binary variable s_i, $i=1,\ldots,n$, each input variable can be considered or not. However, the use of variables s_i is not the single tool to optimize the structure of the SLFN. The configuration of the hidden layer can be adjusted in order to minimize the output error of the model. The activation function $f_j(\cdot)$, $j=1,\ldots,h$, of each hidden node can be either zero, if this neuron is unnecessary, or any (predefined) activation function.

3. Extreme learning machine

The batch ELM was proposed in [6]. In [18] it is proved that a SLFN with randomly chosen weights between the input layer and the hidden layer and adequately chosen output weights are

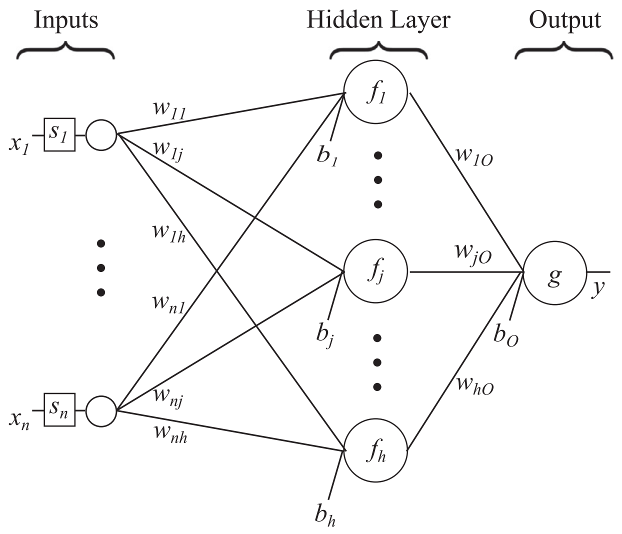


Fig. 1. Single hidden-layer feedforward network with adjustable architecture.

universal approximators with any bounded non-linear piecewise continuous functions.

Considering that N samples are available, the output bias is zero, and the output neuron has a linear activation function, (1) and (2) can be rewritten as

$$\mathbf{y} = (\mathbf{w}_O^T \mathbf{V})^T, \tag{3}$$

where $\mathbf{y} = [y(1), \dots, y(N)]^T$ is the vector of outputs of the SLFN, $\mathbf{w}_O = [w_{1O}, \dots, w_{hO}]^T$ is the vector of output weights, and $\mathbf{V}$ is the matrix of the outputs of the hidden neurons (1) given by

$$\mathbf{V} = \begin{bmatrix} v_1(1) & v_1(2) & \dots & v_1(N) \\ \vdots & \vdots & \ddots & \vdots \\ v_h(1) & v_h(2) & \dots & v_h(N) \end{bmatrix}, \tag{4}$$

with $s_i = 1$, $i = 1, \dots, n$.

Considering that the input weights and bias matrix $\mathbf{W}$,

$$\mathbf{W} = \begin{bmatrix} b_1 & b_2 & \dots & b_h \\ w_{11} & w_{12} & \dots & w_{1h} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nh} \end{bmatrix}, \tag{5}$$

is randomly assigned, the output weights vector $\mathbf{w}_O$ is estimated as

$$\hat{\mathbf{w}}_O = \mathbf{V}^\dagger \mathbf{y}_d, \tag{6}$$

where $\mathbf{V}^\dagger$ is the Moore–Penrose generalized inverse of the hidden-layer output matrix $\mathbf{V}$, and $\mathbf{y}_d = [y_d(1), \dots, y_d(N)]^T$ is the desired output.

Considering that $\mathbf{V} \in \mathbb{R}^{N \times h}$ with $N \geq h$ and $rank(\mathbf{V}) = h$, the Moore–Penrose generalized inverse of $\mathbf{V}$ can be given by

$$\mathbf{V}^\dagger = (\mathbf{V}^T \mathbf{V})^{-1} \mathbf{V}^T. \tag{7}$$

Substituting (7) into (6), the estimation of $\mathbf{w}_O$ can be obtained by the following least-squares solution:

$$\hat{\mathbf{w}}_O = (\mathbf{V}^T \mathbf{V})^{-1} \mathbf{V}^T \mathbf{y}_d. \tag{8}$$

4. Optimized extreme learning machine

In O-ELM, the weights of the output connections are obtained using the same ELM methodology presented in Section 3, however, with a change.

The objective of the least squares method is to obtain the best output weights by solving the following problem:

$$\min(\|\mathbf{y} - \mathbf{y}_d\|_2), \tag{9}$$

where $\|\cdot\|_2$ is the Euclidean norm. The minimum-norm solution to this problem is given by (8).

The use of least squares can be considered as a two-stage minimization problem involving the determination of the solutions to (9), and the determination of the solution with minimum norm among solutions obtained in the previous stage.

The use of Tikhonov's regularization [17] allows the transformation of this two-stage problem into a single-stage minimization problem defined by

$$\min(\|\mathbf{y} - \mathbf{y}_d\|_2 + \alpha \|\mathbf{w}_O\|_2), \tag{10}$$

where $\alpha > 0$ is a regularization parameter.

The solution to this problem is given by [17]

$$\hat{\mathbf{w}}_O = (\mathbf{V}^T \mathbf{V} + \alpha \mathbf{I})^{-1} \mathbf{V}^T \mathbf{y}_d, \tag{11}$$

where $\mathbf{I}$ is the $h \times h$ identity matrix.

If $\mathbf{V}$ is ill-conditioned, problem (10) should be preferred over problem (9) because the solution is numerically more stable [19]. Furthermore, using Tikhonov's regularization, the robustness of the least squares solution against noise is improved.

As previously mentioned, the ELM tends to require more hidden nodes than conventional tuning-based algorithms. Furthermore, the presence of irrelevant variables in the training data set causes a decrease in the performance. To overcome these problems, in O-ELM the determination of the set of input variables, the number and activation function of the neurons in the hidden layer, the connection weights between the inputs and the neurons of the hidden layer, the bias of the hidden layer neurons, and the regularization parameter α is made using an optimization methodology.

The optimization of the SLFN consists in minimizing the following evaluation function:

$$\psi = E_{rmse}(\mathbf{y}, \mathbf{y}_d), \tag{12}$$

where

$$E_{rmse}(\mathbf{y}, \mathbf{y}_d) = \sqrt{\frac{1}{N} \sum_{k=1}^{N} [y(k) - y_d(k)]^2} \tag{13}$$

is the root mean square error (RMSE) between the desired (real) output and the estimated values of the output. To improve the generalization performance, the estimation error $E_{rmse}(\mathbf{y}, \mathbf{y}_d)$ is obtained in a validation data set that has no overlap with the training data set.

In the optimization process, it is considered that the individual/state will be constituted by

$$\mathbf{p}_k = [w_{11}, \dots, w_{nh}, b_1, \dots, b_h, s_1, \dots, s_n, s_1^\lambda, \dots, s_h^\lambda, \alpha]^T;$$

$$k = 1, \dots, m, \tag{14}$$

where $s_j^\lambda \in \{0, 1, 2\}$, $j = 1, \dots, h$, is an integer variable that defines the activation function f_j of each neuron j of the hidden-layer as follows:

$$f_j(\upsilon) = \begin{cases} 0 & \text{if } s_j^\lambda = 0, \\ 1/(1 + \exp(-\upsilon)) & \text{if } s_j^\lambda = 1, \\ \upsilon & \text{if } s_j^\lambda = 2. \end{cases} \tag{15}$$

The use of parameters s_j^λ makes it possible the adjustment of the number of neurons (if $s_j^\lambda = 0$ the neuron is not considered), and the activation function of each neuron (sigmoid or linear function). In this work only these two types of activation function have been used; however, any type of activation function can be considered.

This optimization problem is a problem where the decision variables are a combination of real, integer, and binary variables. So, in order to use any type of optimization method, the approach suggested in [20] was used. The decision variables are mapped into real variables within the interval [0,1] and before computing the evaluation function for each individual, all variables need to be converted into their true value. If the true value of the l-th variable $(l = 1, 2, \dots, v)$ of individual k is real, it is given by

$$z_{k_l} = (z_l^{\max} - z_l^{\min}) p_{k_l} + z_l^{\min}, \tag{16}$$

where $z_l^{\min}$ and $z_l^{\max}$ represent the true variable bounds ($z_l^{\min} \leq z_{k_l} \leq z_l^{\max}$). If it is a integer value,

$$z_{k_l} = rounddown((z_l^{\max} - z_l^{\min} + 1) p_{k_l}) + z_l^{\min}, \tag{17}$$

where $rounddown(\cdot)$ is a function that rounds to the greatest integer that is lower than or equal to its argument. If the true value is binary, it is given by

$$z_{k_l} = round(p_{k_l}), \tag{18}$$

where $round(\cdot)$ is a function that rounds to the nearest integer.

The variables s_i, $i = 1, \dots, n$, are binary variables and thus are converted using (18). The variables s_j^λ, $j = 1, \dots, h$, are integer variables and thus are converted using (17), considering that the lower and upper bounds are 0 and 2, respectively. The input weights w_{ij} and bias b_j are converted using (16), considering that the lower and upper bounds are -1 and 1, respectively. Finally, the regularization parameter is also converted using (16), considering that the lower and upper bounds are 0 and 100, respectively.

The proposed framework can be jointly used with a wide range of optimization methods. In this paper three optimization methods (GA, DE, and SA) were tested, resulting in three learning approaches called genetically O-ELM (GO-ELM), O-ELM by differential evolution (DEO-ELM), and O-ELM by simulated annealing (SAO-ELM). These optimization methods were chosen due to the good performance revealed in several combinatorial optimization problems.

5. Optimization methods

This section presents the three methods used in the optimization of the SLFN.

5.1. Genetic algorithms

The basic principles of GA were introduced by Holland [21]. GAs are population-based search and optimization methods that have been used to solve complex problems effectively [22–24]. The principle of GA is the simulation of the natural processes of evolution applying Darwin's theory of natural selection. In a GA, solutions are encoded into chromosomes (individuals) and the fittest ones are more susceptible to be selected for reproduction, producing offspring with characteristics of both parents.

The GA used in the optimization of the SLFN is based on the approach proposed in [20]. The detailed description of this method is given in this section.

5.1.1. Variable encoding and initial population

As previously mentioned, the optimization of the SLFN is a problem in which the cost function involves real, integer, and binary variables. So, in order to solve this optimization problem all variables are mapped into continuous variables within the interval [0,1].

The initial population $\mathbf{P}$ is randomly chosen with uniform distribution and can be represented by

$$\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}, \tag{19}$$

where

$$\mathbf{p}_k = [p_{k_1}, p_{k_2}, \ldots, p_{k_q}]^T. \tag{20}$$

p_{k_l} is the variable l, $l = 1, 2, \ldots, q$, of chromosome k, $k = 1, 2, \ldots, m$, with $0 \le p_{k_l} \le 1$. m and q are the population size and the number of variables to be tuned, respectively.

5.1.2. Evaluation

Each chromosome represents one possible solution to the optimization problem. Therefore each chromosome can be evaluated using a fitness function that is specific to the problem being solved. As all variables are mapped into continuous values between 0 and 1, before obtaining the fitness of each individual these values need to be converted into the actual variable values using (16)–(18). After this step, the fitness of each individual can be obtained using (12).

5.1.3. Genetic operators

Based upon their fitness values, a set of individuals is selected to survive to the next generation while the remaining are discarded. The surviving individuals become the mating pool and the discarded chromosomes are replaced by new offspring. In this work 40% of the individuals with best fitness survive for the next generation. To select the parents from the mating pool, tournament selection was used. For each parent, three individuals from the mating pool are randomly picked and the individual with best fitness is selected to be a parent. For each pair of parents two new individuals (offspring) are generated by crossover and mutation.

The crossover operation consists in producing offspring from the selected parents. Uniform crossover is used because it generally provides a larger exploration of the search space than other crossover operators [25]. In uniform crossover, first a random binary mask with the same length of the individuals is created. Then, each offspring receives values of variables from the first or second parent depending on whether the value of the mask bit is zero or one: the offspring 1/(2) receives the values from parent 1/(2) if the respective mask bit is one and receives the values from parent 2/(1) if the respective mask bit is zero. Consider the following example:

$$\begin{array}{rcllll}
\text{Parent 1} & = & p_{1_1} & p_{1_2} & p_{1_3} & p_{1_4}, \\
\text{Parent 2} & = & p_{2_1} & p_{2_2} & p_{2_3} & p_{2_4}, \\
\text{Mask} & = & 1 & 0 & 0 & 1 \\
\text{Offspring 1} & = & p_{1_1} & p_{2_2} & p_{2_3} & p_{1_4}, \\
\text{Offspring 2} & = & p_{2_1} & p_{1_2} & p_{1_3} & p_{2_4}.
\end{array}$$

After crossover, the mutation operator is used to maintain the diversity of the population and to prevent the algorithm from being trapped in local minima, thus enabling a thorough exploration of the search space.

For each new offspring a random number r is generated and if $r < r_m$, where r_m is the mutation probability, this offspring is mutated. The mutation used is a two step operator. First, a random element of the individual is replaced by an uniform random value within the interval [0,1]. Being $\mathbf{p}_k = [p_{k_1}, p_{k_2}, p_{k_3}, p_{k_4}]^T$ the offspring, if the second element is selected to be replaced, the mutated chromosome is given by

$$\mathbf{p}_k^1 = [p_{k_1}, p'_{k_2}, p_{k_3}, p_{k_4}]^T, \tag{21}$$

where p'_{k_2} is a new random value within the interval [0,1].

In a second step, a random adjustment factor is added to the chromosome. The adjustment factor comes from multiplying each element l within the previously mutated chromosome $\mathbf{p}_k^1$ by a random number $(-1 \le \beta_{k_l} \le 1)$ and multiplying the resulting chromosome by a mutation factor $(0 \le \eta_k \le 1)$ so that

$$\mathbf{p}_k^c = \eta_k [\beta_{k_1} p_{k_1}, \beta_{k_2} p'_{k_2}, \beta_{k_3} p_{k_3}, \beta_{k_4} p_{k_4}]^T. \tag{22}$$

Finally, the mutated chromosome is given by

$$\mathbf{p}_k^2 = rem(\mathbf{p}_k^1 + \mathbf{p}_k^c), \tag{23}$$

where rem is the remainder of each variable after the division by one.

5.2. Differential evolution

Differential evolution grew out of attempts to solve the Chebychev Polynomial fitting Problem. In 1995 the first publication about DE appeared in a technical report written by Storn and Price [26]. Due to the excellent performance in the 1996 and 1997 International Contest on Evolutionary Optimization in the IEEE International Conference on Evolutionary Computation, DE began to gain prominence in the scientific community [27].

DE is a population-based meta-heuristic approach where new candidate solutions are generated adding a weighted difference vector between two population members to a third member.

Like in GA, in DE the initialization of the population is randomly performed with uniform distribution within the interval [0,1], yielding an initial population with m individuals (19).

After the population initialization and while a stop criterion is not met, for each individual $\mathbf{p}_k$ $(k=1,2,\ldots,m)$ of the population three random individuals (ρ, σ, and τ) are selected, with $\rho \neq \sigma \neq \tau$. The difference vector between two of these individuals is calculated. This difference vector is multiplied by a scaling factor and is added to the third selected individual obtaining the trial vector $\mathbf{p}_k'$:

$$\mathbf{p}_k' = \mathbf{p}_\rho + F(\mathbf{p}_\sigma - \mathbf{p}_\tau), \tag{24}$$

where F is the scaling factor. In order to increase the diversity of the individuals, the following vector $\mathbf{u}_k = [u_{k_1}, u_{k_2}, \ldots, u_{k_q}]^T$ is created, where each element l, $l=1,2,\ldots,q$, is given by

$$u_{k_l} = \begin{cases} p_{k_l}' & \text{if } \mathcal{U}_{[0,1]} \leq C \vee l = \delta_k, \\ p_{k_l} & \text{otherwise,} \end{cases} \tag{25}$$

where $\mathcal{U}_{[0,1]}$ is a uniformly distributed random number within the interval [0,1], C is the acceptance probability, and $\delta_k \in \{1,\ldots,m\}$ is a random index for individual k.

The individual $\mathbf{u}_k$ is selected to substitute $\mathbf{p}_k$ in the population $\mathbf{P}$ if its performance is not worse than the performance of $\mathbf{p}_k$. Like in the GA, before evaluating the performance of an individual its component variables need to be converted into the actual variable values in the application domain using (16)–(18). The fitness of each individual is obtained by the evaluation function (12).

5.3. Simulated annealing

Simulated annealing (SA) is a meta-heuristic proposed in [28] for global optimization problems. It works by emulating the physical process of annealing a solid to obtain a state where the energy configuration is minimal.

Consider the solution $\mathbf{p}_k = [p_{k_1}, p_{k_2}, \ldots, p_{k_q}]^T$ in iteration k. In the SA algorithm, a perturbation is applied to $\mathbf{p}_k$ resulting in a neighbor solution $\mathbf{v}_k$. Considering a minimization problem, this neighbor will be then accepted as the new current solution according to the following acceptance probability C:

$$C = \begin{cases} 1 & \text{if } \psi(\mathbf{v}_k) \leq \psi(\mathbf{p}_k), \\ \exp\left(\dfrac{\psi(\mathbf{v}_k) - \psi(\mathbf{p}_k)}{T_k}\right) & \text{otherwise,} \end{cases} \tag{26}$$

where $\psi(\mathbf{v}_k)$ and $\psi(\mathbf{p}_k)$ are, respectively, the fitness values of $\mathbf{v}_k$ and $\mathbf{p}_k$ as given by (12) and $T_k \in \mathbb{R}^+$ is the temperature in iteration k. Like in the other two optimization methods (GA and DE), before evaluating the performance of a state its component variables need to be converted into the actual variable values in the application domain using (16)–(18). In each iteration the temperature is decreased as $T_{k+1} = \gamma T_k$, where $\gamma \in (0,1)$ is a user-defined constant.

Table 1
Publicly available benchmark data sets.

Dataset	N_t	N_{tt}	n
Boston Housing [29]	253	253	13
Automobile MPG [29]	196	196	6
Cancer [29]	97	97	32
Servo [29]	84	83	4
Price [29]	80	80	15
CPU [29]	105	104	6
Concrete Comp. Strength [29]	515	515	8
Ailerons [30]	7154	6596	40
Pumadyn [31]	4499	3693	32
Elevators [30]	8752	7847	18
Pole Telecommunication [30]	5000	10 000	48
Bank [31]	4500	3692	32
Computer Activity [31]	4096	4096	21
Stock [30]	475	475	9
Communities and Crime [29]	997	997	127
Pyrimidines [32]	37	37	27

6. Results

This section presents experimental results in 16 benchmark data sets. Table 1 presents the number of training samples N_t, the number of testing samples N_{tt}, and the number of input variables of these benchmark data sets. In these data sets, delays between each input variable and the output could be considered and the performance of the methods could possibly be improved [33]; however, as only the learning capability of the SLFN is being

Table 2
Results of the application of the three optimization methods in the optimized extreme learning machine in the benchmark data sets using the RMSE as the performance measure.

Data set	Method	Testing RMSE		Training	Hidden
		Mean	Std	time (s)	neurons
Boston Housing	GO-ELM	**0.3503**	0.0565	4.5463	21.15
	DEO-ELM	0.3530	0.0591	8.3319	21.15
	SAO-ELM	0.3609	0.0648	2.4084	18.45
Automobile MPG	GO-ELM	**0.2608**	0.049	4.3751	21.75
	DEO-ELM	0.2652	0.0522	7.3566	21.75
	SAO-ELM	0.2699	0.0508	1.1807	19.35
Cancer	GO-ELM	**0.5782**	0.0229	4.0442	18.25
	DEO-ELM	0.6317	0.0183	7.4927	17.30
	SAO-ELM	0.6326	0.0224	3.0648	18.50
Servo	GO-ELM	**0.2320**	0.0428	3.6212	21.75
	DEO-ELM	0.2701	0.0248	6.7205	21.75
	SAO-ELM	0.2757	0.0273	0.9686	19.30
Price	GO-ELM	**0.1896**	0.0231	3.7553	18.70
	DEO-ELM	0.2309	0.0662	6.9191	18.70
	SAO-ELM	0.1902	0.0308	1.7578	19.60
CPU	GO-ELM	0.1733	0.0281	3.7444	17.20
	DEO-ELM	0.1730	0.0058	7.2471	17.20
	SAO-ELM	**0.1624**	0.0067	1.1167	18.55
Concrete Comp. Strength	GO-ELM	**0.2738**	0.0230	5.4051	22.05
	DEO-ELM	0.2885	0.0403	9.6447	22.05
	SAO-ELM	0.3104	0.0485	1.7068	19.60
Ailerons	GO-ELM	**0.0912**	0.0007	27.4410	23.55
	DEO-ELM	0.0918	0.0006	45.6752	23.55
	SAO-ELM	0.0946	0.0015	22.4723	20.50
Pumadyn	GO-ELM	**0.1442**	0.0333	17.9558	22.95
	DEO-ELM	0.293	0.0044	28.0148	22.95
	SAO-ELM	0.3074	0.0016	11.5952	21.10
Elevators	GO-ELM	**0.0774**	0.0031	30.3114	24.05
	DEO-ELM	0.0831	0.0087	48.2631	23.65
	SAO-ELM	0.0872	0.0052	12.9927	19.90
Pole Telecommu-nication	GO-ELM	**0.4901**	0.0286	22.0324	25.25
	DEO-ELM	0.5147	0.0122	35.9068	25.25
	SAO-ELM	0.5888	0.0234	20.6218	20.30
Bank	GO-ELM	0.2051	0.0017	16.2897	23.40
	DEO-ELM	**0.2046**	0.0005	29.0433	23.40
	SAO-ELM	0.2117	0.0047	12.0247	20.15
Computer Activity	GO-ELM	**0.0848**	0.0118	17.9445	23.75
	DEO-ELM	0.0926	0.0155	25.3110	23.70
	SAO-ELM	0.1377	0.0230	7.7895	21.25
Stock	GO-ELM	**0.6767**	0.3192	9.5224	23.85
	DEO-ELM	0.7359	0.2155	9.5277	23.85
	SAO-ELM	0.9666	0.3019	1.7721	20.55
Communities and Crime	GO-ELM	**0.2679**	0.0073	9.6412	22.10
	DEO-ELM	0.2729	0.0075	17.2391	22.10
	SAO-ELM	0.2767	0.0079	21.3763	21.15
Pyrimidines	GO-ELM	0.2372	0.0535	3.8585	20.60
	DEO-ELM	**0.2238**	0.0359	7.2562	20.60
	SAO-ELM	0.2376	0.0447	2.6047	19.9

Table 3
Results of the application of the five methods in the benchmark data sets using the RMSE as the performance measure.

Data set	Method	Testing RMSE		Training time (s)	Hidden neurons
		Mean	Std		
Boston Housing	GO-ELM	**0.3503**	0.0565	4.5463	21.15
	ELM	0.4652	0.1901	0.0020	20.00
	IGA-SLFN	0.5208	0.1278	1.6540	28.00
	SaE-ELM	0.4594	0.1628	8.0205	15.00
	LM-SLFN	0.5182	0.1677	1.0935	15.00
Automobile MPG	GO-ELM	**0.2608**	0.0490	4.3751	21.75
	ELM	0.2610	0.0443	0.0025	19.00
	IGA-SLFN	0.4874	0.1582	1.3098	23.00
	SaE-ELM	0.2682	0.0491	7.5290	15.00
	LM-SLFN	0.4030	0.1109	0.7050	17.00
Cancer	GO-ELM	0.5782	0.0229	4.0442	18.25
	ELM	0.5864	0.0240	0.0015	11.00
	IGA-SLFN	**0.5706**	0.1478	2.1523	22.00
	SaE-ELM	0.6169	0.0287	7.6500	15.00
	LM-SLFN	0.8288	0.2090	1.0375	15.00
Servo	GO-ELM	0.2320	0.0428	3.6212	21.75
	ELM	0.2401	0.0263	0.0030	24.00
	IGA-SLFN	0.5316	0.1419	1.0425	15.00
	SaE-ELM	**0.2301**	0.0261	6.6805	15.00
	LM-SLFN	0.2713	0.0566	0.5555	15.00
Price	GO-ELM	**0.1896**	0.0231	3.7553	18.70
	ELM	0.2092	0.0431	<0.0001	10.00
	IGA-SLFN	0.5769	0.1568	1.8273	16.00
	SaE-ELM	0.2853	0.0521	6.7875	15.00
	LM-SLFN	0.4403	0.0807	0.8040	22.00
CPU	GO-ELM	**0.1733**	0.0281	3.7444	17.20
	ELM	0.1772	0.0712	0.0010	11.00
	IGA-SLFN	0.7492	0.3486	1.2839	22.00
	SaE-ELM	0.2449	0.0971	31.8785	19.00
	LM-SLFN	0.3098	0.1069	0.5935	18.00
Concrete Comp. Strength	GO-ELM	**0.2738**	0.0230	5.4051	22.05
	ELM	0.3152	0.0533	0.0025	13.00
	IGA-SLFN	0.7526	0.3365	1.7226	24.00
	SaE-ELM	0.3109	0.0312	38.7825	15.00
	LM-SLFN	0.3765	0.0661	1.5910	17.00
Ailerons	GO-ELM	**0.0912**	0.0007	27.4410	23.55
	ELM	0.0947	0.0019	0.0590	29.00
	IGA-SLFN	0.3809	0.2393	8.3769	15.00
	SaE-ELM	0.0917	0.0006	323.0840	27.00
	LM-SLFN	0.0940	0.0022	24.6795	22.00
Pumadyn	GO-ELM	0.1442	0.0333	17.9558	22.95
	ELM	0.3205	0.0031	0.0360	28.00
	IGA-SLFN	0.3702	0.0354	8.6215	23.00
	SaE-ELM	0.3125	0.0013	237.2725	30.00
	LM-SLFN	**0.0769**	0.0057	45.0315	23.00
Elevators	GO-ELM	0.0774	0.0031	30.3114	24.05
	ELM	0.0882	0.0040	0.0570	26.00
	IGA-SLFN	0.4000	0.1951	11.2353	22.00
	SaE-ELM	0.0776	0.0015	369.6585	27.00
	LM-SLFN	**0.0655**	0.0028	63.2730	15.00
Pole Telecommu-nication	GO-ELM	0.4901	0.0286	22.0324	25.25
	ELM	0.6067	0.0133	0.0410	29.00
	IGA-SLFN	0.9116	0.1000	5.8895	22.00
	SaE-ELM	0.5209	0.0111	264.5200	30.00
	LM-SLFN	**0.1865**	0.0267	49.1555	26.00
Bank	GO-ELM	**0.2051**	0.0017	16.2897	23.40
	ELM	0.6067	0.0133	0.0410	29.00
	IGA-SLFN	0.3171	0.0269	5.5643	15.00
	SaE-ELM	0.5209	0.0111	264.5200	30.00
	LM-SLFN	0.2060	0.0037	14.3920	16.00
Computer Activity	GO-ELM	0.0848	0.0118	17.9445	23.75
	ELM	0.1765	0.0151	0.0340	28.00
	IGA-SLFN	0.5508	0.3076	4.7953	24.00
	SaE-ELM	0.1028	0.0102	209.9595	29.00
	LM-SLFN	**0.0545**	0.0030	9.9655	15.00

Table 3 (*continued*)

Data set	Method	Testing RMSE		Training time (s)	Hidden neurons
		Mean	Std		
Stock	GO-ELM	**0.6767**	0.3192	9.5224	23.85
	ELM	0.7122	0.2705	0.0075	19.00
	IGA-SLFN	0.8848	0.5259	2.2219	16.00
	SaE-ELM	0.6860	0.3295	63.7135	15.00
	LM-SLFN	0.8393	0.4125	1.3660	21.00
Communities and Crime	GO-ELM	**0.2679**	0.0073	9.6412	22.10
	ELM	0.2932	0.0081	0.0110	28.00
	IGA-SLFN	0.4338	0.0528	7.4356	28.00
	SaE-ELM	0.2716	0.0055	92.2715	26.00
	LM-SLFN	0.3454	0.0274	22.0185	15.00
Pyrimidines	GO-ELM	**0.2372**	0.0535	3.8585	20.60
	ELM	0.2642	0.0680	0.0005	11.00
	IGA-SLFN	0.3411	0.1333	1.6119	19.00
	SaE-ELM	0.2399	0.0464	30.6100	19.00
	LM-SLFN	0.3208	0.0818	0.3740	16.00

analyzed, no delay between the input variables and the output variable in all data sets has been considered.

The data was divided as follows: the training and testing sets available in the repositories were used; when these separate data sets are not provided, the first half of the data was used for training, and the second half was used for testing. All the input and output variables have been normalized to the range $[-1,1]$. All simulations have been made in Matlab environment running on a PC with 3.40 GHz CPU with 4 cores and 8 GB RAM.

In order to select which of the three optimization methods will be used in O-ELM, the three approaches GA (GO-ELM), DE (DEO-ELM), and SA (SAO-ELM) were tested in the benchmark data sets of Table 1. There is a Matlab toolbox available online for performing the O-ELM framework, including the three tested optimization methods [34]. The performance of the methods is evaluated using mean and standard deviation of RMSE between the estimated and desired outputs in 20 trials. As in the O-ELM learning methods the structure of the SLFN is optimized, it was considered that the initial number of neurons in the hidden-layer is 30. 30% of the training data set was randomly picked and was used as the validation data set. In GA and DE, the population size was 80 individuals and the maximum number of iterations was 100. In GA, the mutation probability was 10%. In the SA, the maximum number of iterations was 500, the initial temperature was 100 and decreased with $\gamma = 0.95$. As SA is not population-based, a maximum number of iterations five times higher than in GA and DE was considered. In the DE it was considered that F is a random number uniformly distributed in the interval [0.5,2].

Table 2 presents the average of the 20 trials of the three optimization methods in the optimized extreme learning machine in all data sets and the mean number of neurons in the hidden-layer obtained after the optimization. For all data sets, the best RMSE is shown in bold face.

From the analysis of the results it can be verified that the number of neurons in the hidden layer obtained in the three approaches is similar. Concerning training time, in general SAO-ELM is the fastest method and DEO-ELM is the slowest. With respect to fitting performance it can be verified that GO-ELM shows the lowest mean of RMSE in the testing data set in 13 of the 16 benchmark data sets. Therefore the GA was selected as the optimization method to be used in the O-ELM.

In the next step, GO-ELM will be compared with (i) the original batch ELM [6], (ii) the method proposed in [11] (IGA-SLFN), (iii) the self-adaptive evolutionary ELM (SaE-ELM) [16], and (iv) SLFN trained using the Levenberg–Marquardt algorithm (LM-SLFN). In

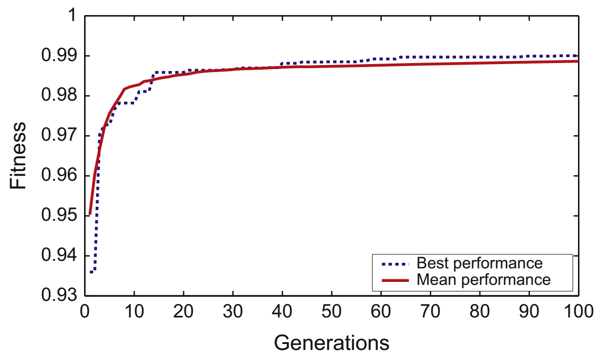


Fig. 2. Convergence of GA in Price data set (trial with best final fitness and mean performances of the 20 trials).

ELM, IGA-SLFN, SaE-ELM, and LM-SLFN the number of neurons in the hidden-layer was gradually increased and the one with the best results in the testing set is presented. Like in the GO-ELM, in the SaE-ELM and LM-SLFN, 30% of the training data set was randomly picked and was used as validation data set. In SaE-ELM, the population size was 80 individuals and the maximum number of generations was 100. In IGA-SLFN, it was considered that the population size was 80 individuals and the maximum number of generations was 500.

Table 3 presents the average of the 20 trials of the five methods in all data sets and the number of neurons in the hidden-layer used to obtain these results. In GO-ELM the mean number of hidden neurons obtained after the 20 optimization trials is presented. Similar to Table 2, the best RMSE is shown in bold face.

From the analysis of the results it can be verified that GO-ELM shows the lowest mean of RMSE in the testing data set in 10 of the 16 benchmark data sets (Housing, Automobile MPG, Price, CPU, Concrete, Ailerons, Bank, Stock, Communities and Crime, and Pyramidines data sets). In the Servo data set, SaE-ELM obtained the lowest mean of RMSE in the testing data set and in the Cancer data set the lowest RMSE was obtained by the IGA-SLFN. The LM-SLFN method obtained the lowest mean of RMSE in the Pumadyn, Elevators, Pole Telecommunication, and Computer Activity data sets. The training time used by GO-ELM is in general larger than in IGA-SLFN, LM-SLFN, and ELM, but GO-ELM also contrasts with these three methods by optimizing not only the parameters but also the structure of the SLFN. In fact, in the learning of the SLFN by the GO-ELM it is only necessary to initialize the methodology with a large number of neurons whereas in the remaining four

Table 4
Relative frequency of selection of input variables by GO-ELM over the 20 trials of each of the Automobile MPG, Servo, Boston Housing, and Price data sets. A value of 1 indicates the variable was selected 20 times.

Data set	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
Boston Housing	0.65	0.50	0.25	0.30	0.55	1.00	0.85	0.90	0.45	0.80	0.95	0.80	0.55	–	–	–	–	–	–	–	–
Automobile MPG	0.40	0.90	0.75	0.95	0.80	0.85	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Servo	0.70	0.80	1.00	0.55	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Price	0.30	0.65	0.85	0.40	0.45	0.25	0.30	0.60	0.45	0.70	0.40	0.40	0.55	0.45	0.40	–	–	–	–	–	–
CPU	0.15	0.95	0.75	0.75	0.70	0.50	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Concrete	0.95	0.85	0.70	1.00	0.85	0.60	0.90	1.00	–	–	–	–	–	–	–	–	–	–	–	–	–
Elevators	1.00	0.25	1.00	0.55	0.20	1.00	0.60	1.00	0.25	0.90	0.80	0.85	0.90	0.20	0.55	0.70	0.65	0.80	–	–	–
Computer Activity	0.50	0.20	0.55	0.40	0.60	0.50	0.40	0.00	0.10	0.05	0.10	0.20	0.20	0.10	0.10	0.20	0.40	0.55	0.35	0.25	1.00
Stock	1.00	1.00	0.90	0.85	1.00	1.00	1.00	0.55	0.90	–	–	–	–	–	–	–	–	–	–	–	–

methods several tests have to be made to select the optimal number of hidden-layer neurons, thus compensating the larger training time of the GO-ELM. The lowest training time was obtained by the ELM as expected.

Fig. 2 presents the GA convergence curves of GO-ELM in the Price data set. It can be seen that the convergence of the GA is fast and therefore, in applications where a short training time is crucial, the training time of the GO-ELM can be shortened if necessary. However, this can cause a reduction of the GO-ELM performance.

Table 4 shows the relative frequency of the presence of each input variable in the set of selected input variables obtained after each trial by GO-ELM in the Boston Housing, Automobile MPG, Servo, Price, CPU, Concrete, Elevators, Computer Activity, and Stock data sets.

From the analysis of the table it is clear that in these data sets some input variables were selected with much more frequency than others and therefore it can be concluded that some irrelevant input variables exist and a mechanism to select the input variables should be used. Due to the large number of input variables in Cancer, Ailerons, Pumadyn, Pole Telecommunication, Bank, Communities and Crime, and Pyramidines data sets, the results of GO-ELM with respect to the most selected input variables are not presented; however, on average in each trial a set with 12.75 input variables was obtained in Cancer data set (39.84% of total input variables), 20.55 (51.38% of total input variables) in Ailerons data set, 2.80 (8.75% of total input variables) in Pumadyn data set, 22.20 (46.25% of total input variables) in Pole Telecommunication data set, 13.25 (41.41% of total input variables) in Bank data set, 59.50 (46.85% of total input variables) in Communities and Crime data set, and 12.45 (46.11% of total input variables) in Pyramidines data set.

7. Conclusion

A novel learning framework for SLFNs called optimized extreme learning machine was presented. Like in the original ELM, the output weights are obtained using the least squares algorithm, but with Tikhonov's regularization. The proposed integration of regularization penalizes solutions with larger norms and allows an improvement in the SLFN generalization capability, improving the performance in the test data. In order to solve the tendency of ELM to require more neurons in the hidden-layer than conventional tuning-based learning algorithms and the reduction of performance exhibited by ELM in the presence of irrelevant input variables, in the O-ELM framework the structure and the parameters of the SLFN are selected using an optimization method. From a wide range of available optimization methods, GA, DE, and SA were selected to test the proposed framework. These methods were selected due to the good performance revealed in several combinatorial optimization problems. It was concluded that GA is the optimization method that jointly with the proposed framework shows the best performance.

To validate and demonstrate the performance and effectiveness of the O-ELM framework, it was applied on 16 benchmark data sets available in public repositories. The performance of the proposed framework using GA was better than the performance of the IGA-SLFN, SaE-ELM, LM-SLFN, and ELM in 10 of the 16 data sets. The results also show that in the proposed framework the number of neurons in the hidden-layer does not need to be selected by trial-and-error and the relevant input variables can be automatically selected, thus reducing the network size and improving the generalization capability.

Acknowledgments

This work was supported by Project FAir-Control “Factory Air Pollution Control” (reference: E!6498), supported by the Eurostars Programme of the EUREKA network, financed by “Fundação para a Ciância e a Tecnologia” (FCT), “Agência de Inovação” (AdI), and the Seventh Framework Programme for Research and Technological Development (FP7) of the European Union.

Tiago Matias and Rui Araújo acknowledge the support of FCT project PEst-C/EEI/UI0048/2011.

Francisco Souza has been supported by FCT under Grant SFRH/BD/63454/2009.

C. H. Antunes acknowledges the support of FCT project PEst-C/EEI/UI0308/2011 and QREN Mais Centro Program iCIS project (CENTRO-07-ST24-FEDER-002003).

References

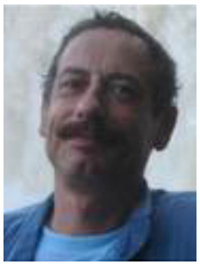
[1] J. Deng, K. Li, G.W. Irwin, Fast automatic two-stage nonlinear model identification based on the extreme learning machine, Neurocomputing 74 (16) (2011) 2422–2429 http://dx.doi.org/10.1016/j.neucom.2010.11.035.
[2] B. Subudhi, D. Jena, Nonlinear system identification using memetic differential evolution trained neural networks, Neurocomputing 74 (10) (2011) 1696–1709 http://dx.doi.org/10.1016/j.neucom.2011.02.006.
[3] R. Hecht-Nielsen, Theory of the back propagation neural network, in: International Joint Conference on Neural Networks, 1989, pp. 593–605. http://dx.doi.org/10.1109/IJCNN.1989.118638.
[4] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[5] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, Op-elm: optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162 http://dx.doi.org/10.1109/TNN.2009.2036259.
[6] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501 http://dx.doi.org/10.1016/j.neucom.2005.12.126.
[7] S. Chen, E.S. Chng, K. Alkadhimi, Regularized orthogonal least squares algorithm for constructing radial basis function networks, Int. J. Control 64 (5) (1996) 829–837.

[8] K.P. Ferentinos, Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms, Neural Netw. 18 (7) (2005) 934–950 http://dx.doi.org/10.1016/j.neunet.2005.03.010.
[9] B. Subudhi, D. Jena, Differential evolution and levenberg marquardt trained neural network scheme for nonlinear system identification, Neural Process. Lett. 27 (3) (2008) 285–296 http://dx.doi.org/10.1007/s11063-008-9077-x.
[10] P. A. C. Valdivieso, J. J. M. Guervós, J. González, V.M.R. Santos, G. Romero, Saprop: optimization of multilayer perceptron parameters using simulated annealing, in: Lecture Notes in Computer Science, vol. 1606, Springer, 1999, pp. 661–670.
[11] F.H.F. Leung, H.K. Lam, S.H. Ling, P.K.S. Tam, Tuning of the structure and parameters of neural network using an improved genetic algorithm, IEEE Trans. Neural Netw. 14 (1) (2003) 79–88.
[12] J.-T. Tsai, J.-H. Chou, T.-K. Liu, Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm, IEEE Trans. Neural Netw. 17 (1) (2006) 69–80.
[13] J.-T. Tsai, T.-K. Liu, J.-H. Chou, Hybrid Taguchi-genetic algorithm for global numerical optimization, IEEE Trans. Evol. Comput. 8 (4) (2004) 365–377 http://dx.doi.org/10.1109/TEVC.2004.826895.
[14] J. Xu, Y. Lu, D.W.C. Ho, A combined genetic algorithm and orthogonal transformation for designing feedforward neural networks, in: 3rd International Conference on Natural Computation, vol. 1, 2007, pp. 10–14. http://dx.doi.org/10.1109/ICNC.2007.13.
[15] Q.-Y. Zhu, A. Qin, P. Suganthan, G.-B. Huang, Evolutionary extreme learning machine, Pattern Recogn. 38 (10) (2005) 1759–1763 http://dx.doi.org/10.1016/j.patcog.2005.03.028.
[16] J. Cao, Z. Lin, G.-B. Huang, Self-adaptive evolutionary extreme learning machine, Neural Process. Lett. 36 (3) (2012) 285–305 http://dx.doi.org/10.1007/s11063-012-9236-y.
[17] J. Honerkamp, J. Weese, Tikhonov's regularization method for ill-posed problems, Contin. Mech. Thermodyn. 2 (1990) 17–30, http://dx.doi.org/10.1007/BF01170953.
[18] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892 http://dx.doi.org/10.1109/TNN.2006.875977.
[19] A. Ben-Israel, T.N. Greville, Generalized Inverses: Theory and Applications, 2nd ed., Springer-Verlag, New York, USA, 2003.
[20] R.L. Haupt, Antenna design with a mixed integer genetic algorithm, IEEE Trans. Antennas Propag. 55 (3) (2007) 577–582 http://dx.doi.org/10.1109/TAP.2007.891510.
[21] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, USA, 1975.
[22] M.H. Mohamed, Rules extraction from constructively trained neural networks based on genetic algorithms, Neurocomputing 74 (17) (2011) 3180–3192 http://dx.doi.org/10.1016/j.neucom.2011.04.009.
[23] F. Souza, T. Matias, R. Araújo, Co-evolutionary genetic multilayer perceptron for feature selection and model design, in: IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA 2011), 2011, pp. 1–7. http://dx.doi.org/10.1109/ETFA.2011.6059084.
[24] C. Bi, Deterministic local alignment methods improved by a simple genetic algorithm, Neurocomputing 73 (13–15) (2010) 2394–2406 http://dx.doi.org/10.1016/j.neucom.2010.01.023.
[25] R.L. Haupt, S.E. Haupt, Practical Genetic Algorithms, 2nd ed. with CD-ROM, John Wiley & Sons, New York, NY, 2004.
[26] R. Storn, K. Price, Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, Technical Report, International Computer Science Institute (ICSI), Berkeley, California, 1995.
[27] V. Feoktistov, Differential Evolution: In Search of Solutions, Springer-Verlag, New York Inc., Secaucus, NJ, USA, 2006.
[28] S. Kirkpatrick, C.D. Gelatt, Jr., M. P. Vecchi, Optimization by Simulated Annealing, Technical Report, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1982.
[29] A. Frank, A. Asuncion, UCI machine Learning Repository. URL: ⟨http://archive.ics.uci.edu/ml⟩, 2010.
[30] L. Torgo, URL: ⟨http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html⟩.
[31] DELVE Repository by University of Toronto. URL: ⟨http://www.cs.toronto.edu/~delve/data/datasets.html⟩.
[32] R.D. King, S. Muggleton, R.A. Lewis, M.J. Sternberg, Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase, in: Proceedings of the National Academy of Sciences of the United States of America, vol. 89, 1992, pp. 11322–11326.
[33] M. Ragulskis, K. Lukoseviciute, Non-uniform attractor embedding for time series forecasting by fuzzy inference systems, Neurocomputing 72 (10–12) (2009) 2618–2626 http://dx.doi.org/10.1016/j.neucom.2008.10.010.
[34] T. Matias, F. Souza, R. Araújo, The O-ELM Toolbox. URL: ⟨http://www.isr.uc.pt/~tmatias/⟩, 2013.

Tiago Matias received his B.Sc. and M.Sc. degrees in Electrical and Computer Engineering (Automation branch) from the University of Coimbra, in 2011. He is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at the University of Coimbra. Since 2011, he is a Researcher at the "Institute for Systems and Robotics - University of Coimbra" (ISR-UC). His research interests include optimization, meta-heuristics, and computational intelligence.

Francisco Souza was born in Fortaleza, Ceará, Brazil, 1986. He received the B.Sc. degree in Electrical Engineering (Automation branch) from the University Federal of Ceará, Brazil. He is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at the University of Coimbra. Since 2009, he is a Researcher at the "Institute for Systems and Robotics - University of Coimbra" (ISR-UC). His research interests include machine learning and pattern recognition with application to industrial processes.

Rui Araújo received the B.Sc. degree (Licenciatura) in Electrical Engineering, the M.Sc. degree in Systems and Automation, and the Ph.D. degree in Electrical Engineering from the University of Coimbra, Portugal, in 1991, 1994, and 2000, respectively. He joined the Department of Electrical and Computer Engineering of the University of Coimbra where he is currently an Assistant Professor. He is a founding member of the Portuguese Institute for Systems and Robotics (ISR-Coimbra), where he is now a researcher. His research interests include computational intelligence, intelligent control, computational learning, fuzzy systems, neural networks, estimation, control, robotics, mobile robotics and intelligent vehicles, robot manipulators control, sensing, soft sensors, automation, industrial systems, embedded systems, real-time systems, and in general architectures and systems for controlling robot manipulators, mobile robots, intelligent vehicles, and industrial systems.



Carlos Henggeler Antunes received his Ph.D. degree in Electrical Engineering (Optimization and Systems Theory) from the University of Coimbra, Portugal, in 1992. He is a full professor at the Department of Electrical and Computer Engineering, University of Coimbra. His research interests include multiple objective optimization, meta-heuristics, and energy planning, namely demand-responsive systems.