Neurocomputing ■ (■■■■) ■■■–■■■

Contents lists available at ScienceDirect

Neurocomputing

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

μG2-ELM: An upgraded implementation of μ G-ELM

B. Lacruz, D. Lahoz, P.M. Mateo

Depto. Métodos Estadísticos, Universidad de Zaragoza, Spain

ARTICLE INFO

Article history:
Received 28 January 2015
Received in revised form
21 July 2015
Accepted 23 July 2015
Communicated by G.-B. Huang

Keywords:
Artificial neural networks
Extreme Learning Machines
Multi-objective evolutionary algorithms
Decision theory

ABSTRACT

μG-ELM is a multiobjective evolutionary algorithm which looks for the best (in terms of the MSE) and most compact artificial neural network using the ELM methodology. In this work we present the μG2-ELM, an upgraded version of μG-ELM, previously presented by the authors. The upgrading is based on three key elements: a specifically designed approach for the initialization of the weights of the initial artificial neural networks, the introduction of a re-sowing process when selecting the population to be evolved and a change of the process used to modify the weights of the artificial neural networks. To test our proposal we consider several state-of-the-art Extreme Learning Machine (ELM) algorithms and we confront them using a wide and well-known set of continuous, regression and classification problems. From the conducted experiments it is proved that the μG2-ELM shows a better general performance than the previous version and also than other competitors. Therefore, we can guess that the combination of evolutionary algorithms with the ELM methodology is a promising subject of study since both together allow for the design of better training algorithms for artificial neural networks.

1. Introduction

Artificial neural networks (ANNs) are a well-known tool which have allowed us to model and solve a high amount of real problems in many different fields: healthcare [1,2], image processing [3], control systems [4], data mining (pattern recognition, clustering, feature detection) [5], civil engineering [6], business [7], stock market predictions [8], etc. Many algorithms have been developed to deal with them [9] and we center our attention in Extreme Learning Machines (ELMs) together with Evolutionary Algorithms (EAs).

ELMs are a set of techniques for learning generalized single-hidden layer feedforward neural networks (SLFNs) introduced in [10] in 2004. Its computer-based learning efficiency, and its formalizing and universal approximation capability are proved in [10,11,13]. The basic ELM consists of starting with a SLFN with a high enough number of hidden nodes, then assigning at random its hidden weights and biases and, finally, using a simple calculation (see Section 2), obtaining the remaining weights. This simple process provides algorithms which need little human intervention in tuning control parameters. Also they show high learning speed and, as it is verified by means of its application, high generalization capability. These features have made ELMs so interesting for researchers. A fast search of the words “extreme”, “learning” and “machine” in Google Scholar provides more than four hundred and fifty thousand entries. A similar search in Scopus©, restricted to articles appeared in journals, provides around nine hundred results, being, at this moment, Neurocomputing and Neural Computing and Applications the journals which more promote the ELM methodology, providing almost 25% of the publications.

A large amount of variants of the original batch learning ELM can be found in the literature as well as many interesting applications to real problems. See, for example, [12,14–16] and references therein. In this work we consider some of the most representative versions used for traditional classification and regression tasks and which take into account different aspects of the technique: (i) the training data processing: OS-ELM [17] and EOS-ELM [18], (ii) the optimization of the network architecture: EI-ELM [13], OP-ELM [20] and EM-ELM [21], (iii) the generation of the input weights and biases: SaE-ELM [23] and (iv) the structure of the data set: W-ELM [24]. All of them will be briefly comment in Section 5.

On the other hand, EAs are search and optimization methods based on the principles of natural evolution and genetics that try to approximate the optimal solution of a given problem [25]. The EAs, in both the single and the multi-objective cases [26], have shown a great potential during the last two decades. For the multi-objective case, which we are dealing with, an updated list of references with close to 9500 items is maintained by C.A. Coello in [27].

In this paper we develop an upgraded version of the micro-genetic algorithm, μG-ELM, we introduced in [28]. This μG-ELM is a bi-objective evolutionary algorithm which is able to decide in only one run the appropriate number of hidden nodes as well as the corresponding weights and biases for minimizing the training error. Then, it belongs to the ELM groups (ii) and (iii) described above. It incorporates a regression device to determine if the solutions search must be conducted to look for populations containing bigger (with more hidden nodes), smaller (with less

http://dx.doi.org/10.1016/j.neucom.2015.07.069
0925-2312/

hidden nodes) or similar (with a similar number of hidden nodes) SLFNs. In addition, we designed a specific mutation operator for evolving the number of hidden nodes and a survival selection operator. This algorithm was tested against a set of ELM algorithms and a classical BP algorithm in [28] and it showed significant better performance.

Other advantages of μG-ELM, besides those mentioned above, are that, in general, as well as any other evolutionary algorithm, is stable and robust, since bad solutions survive with small probability; and, over-fitting is avoided by searching for compact networks. However, we observed that for some data sets the algorithm seemed to exhibit some premature convergence problems because the elements of the generated populations tended quickly to be very similar. On the other hand, we also realized that depending on the initial values of the weights and biases of the networks in the initial population, the resultant solutions were substantially different in some data sets. This is a well-known and important fact and many studies can be found related to the initialization of weights, for instance [29–35,37,38].

Due to these facts, in this work we tackle a set of modifications focused in two directions on: first increasing the exploratory capability of the algorithm and second, building appropriate initial solutions by looking for the appropriate initial weights and biases for the networks.

For the first task, our proposal is the use of a Gaussian mutation operator, for substituting the polynomial mutation operator previously used [39], which allows us to extend the range of possible values for the mutated weights and biases. In addition, the Gaussian distribution will be also used in all the steps in which the original algorithm used the uniform distribution. Furthermore, we have introduced a re-sowing process which serves to add at the end of each iteration some solutions to the population to be evolved obtained from scratch and since this new device tends to reduce the number of hidden nodes in the elements of the final population, we propose a small modification of the mutation of the number of hidden nodes process to widen their range and to compensate the reduction.

For the second task, a new pre-process based on decision theory is incorporated in order to initialize the networks of the initial population. It takes into account the special features of the problem being modeled and it is fast enough to not increasing substantially the computational time for training the neural networks. It is worthy to notice that this pre-process can be used in the initialization of any other kind of learning algorithms.

The paper is organized as follows: in Section 2 we present the previous establishments and notations for the representation of SLFNs, the ELM methodology and our bi-objective problem. Also the original μG-ELM algorithm is briefly described. In Section 3 the four modifications of the algorithm are presented. Section 4 is devoted to tune our algorithm. Two experiments are developed to determine the best version of the new μG2-ELM algorithm. In Section 5 we compare our μG2-ELM under the best implementation selected as a result of the two previous experiments with the old version and with several competitors. Summary and concluding remarks are provided in Section 6.

2. Brief on ELM, multi-objective optimization problems and the μG-ELM

In this section we present the basic elements and notations for understanding the ELM methodology and the bi-objective optimization problem used to guide the search of the EA. Also we briefly describe our μG-ELM.

2.1. Single hidden layer feedforward networks and the ELM methodology

A single hidden layer feedforward network with activation function $g(\cdot)$ satisfying certain mild conditions [19], linear output function, n_1 input variables, n_2 hidden nodes and n_3 output variables can be represented by

$$f_r(\vec{x}) = \sum_{j=1}^{n_2} \beta_{jr} g\left(\sum_{i=1}^{n_1} a_{ij}x_i + b_j\right), \tag{1}$$

where a_{ij}, b_j and β_{jr}, $i=1,\ldots n_1$, $j=1,\ldots,n_2$, $r=1,\ldots,n_3$ are the weights and biases of the layers 1 (hidden) and the weights of the layer 2 (output), respectively. In the rest of the paper we will use weights to refer to the weights and biases.

Now, given a data set with p different patterns, denoted by means of

$$(\vec{x}_s, \vec{t}_s) = (x_{s1},\ldots,x_{sn_1}, t_{s1},\ldots,t_{sn_3}), \quad s=1,\ldots,p, \tag{2}$$

where $\vec{x}_s$ corresponds to the input variables and $\vec{t}_s$ to the target or result, the ELM methodology proposes to set at random the values of the components of $\vec{a}$ and $\vec{b}$ which correspond to the hidden weights a_{ij} and b_j, $i=1,\ldots,n_1, j=1,\ldots,n_2$ and then, to obtain the matrix of output weights, $\beta=(\beta_{jr}), j=1,\ldots, n_2, r=1,\ldots,n_3$, by means of the calculation

$$\beta = \mathbf{H}^{\dagger}\mathbf{T}, \tag{3}$$

where $\mathbf{T}=(\vec{t}_s), s=1,\ldots,p$, is the vector of targets and $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse of matrix $\mathbf{H}=(g(\sum_{i=1}^{n_1} a_{ij}x_{si}+b_j), s=1,\ldots,p, j=1,\ldots,n_2$. Then, Eq. (1) can be used to obtain the response of the network that minimizes the mean square error

$$MSE(\vec{a},\vec{b}) = \frac{1}{p}\sum_{s=1}^{p}\sum_{r=1}^{n_3}(t_{sr}-f_r(\vec{x}_s))^2. \tag{4}$$

2.2. The bi-objective optimization problem and its solutions

The aim of the μG-ELM algorithm is to find the SLFN which simultaneously minimizes the MSE and the number of hidden nodes. For that purpose we state the following bi-objective optimization problem

$$\begin{aligned} &\min \vec{F}(\vec{a},\vec{b},n_2) = (MSE(\vec{a},\vec{b}),n_2)\\ &\text{s. t. : } (\vec{a},\vec{b}) \in \mathcal{R}^{n_2\times(n_1+1)}\\ &1 \le n_2 \le n_{max}. \end{aligned} \tag{5}$$

Note that we only include the parameters $(\vec{a},\vec{b})$ and n_2 since Eq. (3) provides us the remaining parameters of the SLFN $\beta=(\beta_{jr}), j=1,\ldots,n_2, r=1,\ldots,n_3$.

The main difference between single and multi-objective optimization lies in the fact that, in general, multi-objective optimization problems have not a solution which simultaneously minimizes all the objectives. Because of this, it is usual to look for a set of solutions, called Pareto optimal set (or set of efficient solutions), which verifies that there is no other feasible solution which strictly improves one component of its objective function without worsening at least one of the remaining ones.

For our problem in Eq. (5) the Pareto optimal solutions are the ones for which there is no other solution with an strictly smaller mean square error and lower than or equal number of hidden nodes.

Fig. 1 shows the MSE versus the number of hidden nodes of a population of 35 solutions where for each solution represented with a triangle, there is at least one solution represented with a

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

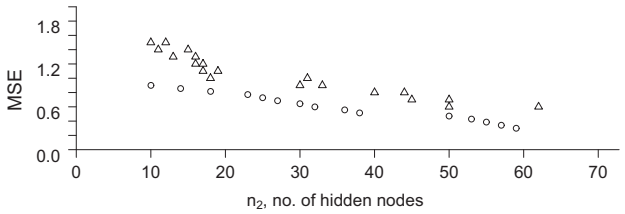

Fig. 1. MSE versus the number of hidden nodes of a population of 35 SLFNs. Circles represent efficient solutions.

circle with smaller MSE and smaller or equal number of hidden nodes. Then, circles represent the set of Pareto optimal solutions.

2.3. The μG-ELM algorithm

Procedure 1 shows the schema of our μG-ELM algorithm which is described in detail in [28]. First, an initial population (set of vectors $(\vec{a}, \vec{b}, n_2)$) is built. Then, the remaining parameters of each SLFN are obtained using Eq. (3) and the MSE is calculated on training and test sets by means of Eq. (4). After that, a regression device, which relates their MSE computed on the test set and their number of hidden nodes, decides whether the algorithm has to search for networks with more, less or similar number of hidden neurons, by setting the values of the variables *add* and *subtract*. If $add == TRUE$, the algorithm looks for bigger networks, if $subtract == TRUE$ for smaller ones and, when both variables are *TRUE* the algorithm looks for networks with a number of hidden neurons around a certain value [28, Fig. 3].

Next, an iterative process is repeated. A small subset of solutions is selected from the whole population and for which reason the algorithm is called micro-genetic. Their number of hidden nodes is mutated taking into account the decision of the regression device. Eq. (3) is applied to complete the SLFNs and their MSE is computed on the training set. Then, the hidden weights of the new solutions are evolved in the micro-loop and the regression device is applied again to determine whether the algorithms will look for bigger, similar or smaller networks in the following iteration. Finally, the current population and the evolved solutions in the micro-loop are joined and a new population is obtained by means of a specific selection process which also takes into consideration the value of the variables *add* and *subtract*.

Procedure 1. The original μG-ELM algorithm. * at the beginning of a line indicates that this step will be modified in this paper.

1: (*) Build the initial population P^1 and set $Q^1 = \varnothing$
2: Evaluate P^1 on training and test sets
3: Set *add* and *subtract* variables (Regression device)
4: **for** $t = 1$ **to** *no_of_iterations* **do**
5: Select Q^t from P^t to be evolved
6: (*)Mutate the number of hidden nodes of Q^t
7: Evaluate Q^t on training set
8: (*)The micro-loop. Evolving Q^t
9: Set *add* and *subtract* variables (Regression device)
10: (*)Select P^{t+1} from $P^t \cup Q^t$
11: **end for**

3. Modifications to improve the μG-ELM

The first modification is referred to the use of the Gaussian distribution instead of the uniform for setting the weights of the hidden neurons in several steps of our algorithm. It is also used in the polynomial mutation operator applied in the micro-loop. The second modification is the introduction of a re-sowing process to complete the population used to start the next iteration with a small modification of the process used to mutate the number of hidden nodes. These changes increase in some extent the exploratory capacity of the algorithm. The third modification consists of outlining a decision problem to select the appropriate initial weights.

3.1. The use of a Gaussian distribution for setting the weights and biases

In the original μG-ELM we use the uniform distribution in several steps of the process shown in Procedure 1. In Line 1, the weights of the hidden nodes of the initial population are randomly generated from $U[-1,1]$. In Line 6, when the regression device decides neither to increase nor to decrease the number of hidden nodes, new individuals are generated and the weights of the hidden nodes are also randomly generated from $U[-1,1]$. Furthermore, when the regression device decides to increase the number of hidden nodes, we select at random a hidden node common to all the individuals in the current population, the minimum and maximum values of all their components are calculated and then, the algorithm generates the new weights from a uniform distribution between these minimum and maximum values. These values are also used to compute the maximum range of variation for the polynomial mutation operator [39] used in the micro-loop in Line 8. Therefore, the values of the weights were always restricted by the extremes of the Uniform distribution when sometimes it could be interesting go far away these limits to obtain potentially better solutions. In the new version all these processes are accomplished using the Gaussian distribution $N(\mu,\sigma)$, which allows the parameters to vary in $(-2\sigma, 2\sigma)$ with probability equal to 0.96 but it also permits us to obtain values outside this interval with a probability greater than 0. Therefore, on one hand, we obtain a more stable process to set the values of the weights, since it is most probably that they take values around the mean μ (with higher probability as smaller the standard deviation σ is), and on the other hand, we allow the process for searching outside the limits established by the weights in the current population.

In addition, the Gaussian distribution is also used in the re-sowing device (Section 3.2) that we include in the final selection process (Line 10). There, new solutions have to be generated and their corresponding new hidden weights have to be assigned. So, this choice also simplifies and unifies our algorithm since in the original version we used Uniform distributions in some steps and polynomial distribution in the weights and bias mutation operator.

The use of the Gaussian distribution is implemented as follows. The weights of the hidden nodes of the elements of the initial population (Line 1) are generated from a $N(0,\sigma_0)$. The value of σ_0 is established as we explain in Section 3.2. For the perturbations of the weights in the micro-loop, we use a $N(0,\sigma_j)$, $j = 1,\ldots,n_1+1$ (the n_1 hidden weights and the bias). In the remaining situations we use a $N(\mu_j,\sigma_j), j = 1,\ldots,n_1+1$, where the values of μ_j and σ_j are estimated at the beginning of each iteration from the values of all associated input-to-hidden weights and biases of a randomly selected hidden node common to all individuals in the population.

3.2. The re-sowing process

The main aim of this re-sowing process is to introduce new information in the population to be evolved in order to increase, in some extent, the exploratory capability of the algorithm without loosing the previous interest in obtaining solutions with the appropriate number of hidden nodes, which is established by the regression device.

In the original version of the μG-ELM, at the end of each iteration, the process builds the new population P^{t+1} by applying a selection process to the elements of the current population, P^t,

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

together with a small set of new solutions, Q^t, evolved in the micro-evolutionary loop. This selection process is based on the ranking proposed by Goldberg [40, p. 201] and the decision of the regression device. Then, this new population is used to start the next iteration. In the new proposal, we obtain a percentage α of the population with the re-sowing process and the remaining $(1-\alpha)\%$ of the population with the mentioned process.

The re-sowing process is done in two steps. First the number of hidden nodes of the new solutions n_2 is set taking into consideration if the algorithm is looking for bigger or smaller networks. This step is shown in Procedure 2. The second step consists of assigning the weights to the new networks. It is done using the normal distribution as we have explained above in Section 3.1.

After this process the solutions are incorporated to the population and evaluated using the training and test sets in order to start a new iteration of the algorithm.

Procedure 2. Assigning the number of hidden nodes for the new elements in the re-sowing process. V corresponds to the x coordinate of the vertex of the parabola fitted in the regression device [28, Section 4] and LN and UN are the minimum and maximum number of hidden nodes in P^t, the current population.

$R = (UN - LN)$
$R_{Midpoint} = (UN + LN)/2$
if $add == TRUE$ **and** $subtract == FALSE$ **then**
  Select n_2 at random in $[R_{Midpoint}, UN + R/2]$
else if $subtract == TRUE$ **and** $add == FALSE$ **then**
  Select n_2 at random in $[LN - R/2, R_{Midpoint}]$
else if $add == subtract == TRUE$ **then**
  Select n_2 at random in $[V \pm R/6]$
else
  Select n_2 at random in $[LN, UN]$
end if

3.3. Mutation of the number of hidden nodes

The resowing process tends to reduce the range of variation of the number of hidden nodes since we select the $\alpha\%$ of the biggest or smallest SLFNs in $P^t \cup Q^t$, depending on the decision of the regression device. Fig. 2 shows the MSE versus the number of hidden nodes of 15 efficient solutions in $P^t \cup Q^t$ for which their MSE decreases when the number of hidden nodes increases, then the regression device would decide to increase the number of nodes. In these conditions, and assuming that the size of the population is for example equal to 10, the μG-ELM would select 10 solutions (Fig. 2, center) and the μG2-ELM with $\alpha = 0.2$ would select 8 and, in both cases, those with the highest number of hidden nodes. Therefore, the range of the number of hidden nodes is smaller in the second case. Then, in order to compensate this reduction we propose a minor modification of the number of hidden nodes mutation operator.

In the original mutation process of the number of nodes, the maximum variation depended on the range of variation of the number of hidden nodes of the individuals in the current population and on the current number of hidden nodes of the individual to be mutated. The process is shown in Procedure 3 with lines labeled with 'Old'. In the new proposal this process has been simplified and only depends on the range of variation of the number of hidden nodes (Procedure 3 with lines labeled with 'New'). It is worthy to observe that the new perturbation is always equal to or higher than the old one. For instance, when $add == TRUE$ and $subtract == FALSE$, $0.5 \cdot R$ is equal to $0.25 \cdot (R + UN - n_2) = 0.5 \cdot R - 0.25(n_2 - LN)$ when the number of hidden nodes n_2 is equal to LN and bigger otherwise.

Procedure 3. Mutation of the number of hidden nodes. u is a uniform random value in $(0, 1)$, $\lceil \cdot \rceil$ represents the ceil function, V, LN, UN and P^t are as in Procedure 2. Lines labeled with -Old- correspond to the original version of the algorithm and those labeled with -New- correspond to the new proposal.

$R = UN - LN$
if $add == TRUE$ **and** $subtract == FALSE$ **then**
  Old: $n_2 = n_2 + \lceil u \cdot \min\{0.25(R + UN - n_2), n_{max} - n_2\} \rceil$
  New: $n_2 = n_2 + \lceil u \cdot \min\{0.5 \cdot R, n_{max} - n_2\} \rceil$
else if $subtract == TRUE$ **and** $add == FALSE$ **then**
  Old: $n_2 = n_2 - \lceil u \cdot \min\{0.25(R + n_2 - LN), n_2 - 1\} \rceil$
  New: $n_2 = n_2 - \lceil u \cdot \min\{0.5 \cdot R, n_2 - 1\} \rceil$
else if $add == subtract == TRUE$ **then**
  $n_2 = \lceil V + (1 - 2 \cdot u)(R/6) \rceil$
else
  $n_2 = n_2$
end if

3.4. Initial weights

The third important incorporation is the design of an initial pre-process to determine the range of variation in which the weights of the elements of the initial population have to be selected. It is well-known that it is important how the initial weights are chosen in order to apply different kinds of ANN training algorithms. Several works can be found in the literature devoted to this topic. For instance, in [29] a classical and well-known method to initialize the weights biases is proposed. The study in [30] shows that for large neural networks for pattern classification problems, the generalization performance is improved by selecting small weights. In [31] their authors propose two methods to initialize the parameters of feedforward wavelet networks, a simple heuristic and another based on gradient techniques. In [32] and [36] several initialization methods of weights are studied for back propagation learning algorithms and ELMs, respectively. The authors of [33] propose a fast training algorithm which can be used also to initialize the parameters of a neural network, based on sensitivity analysis. Rahnamayan et al. [34] propose the opposition-based learning method which, unlike

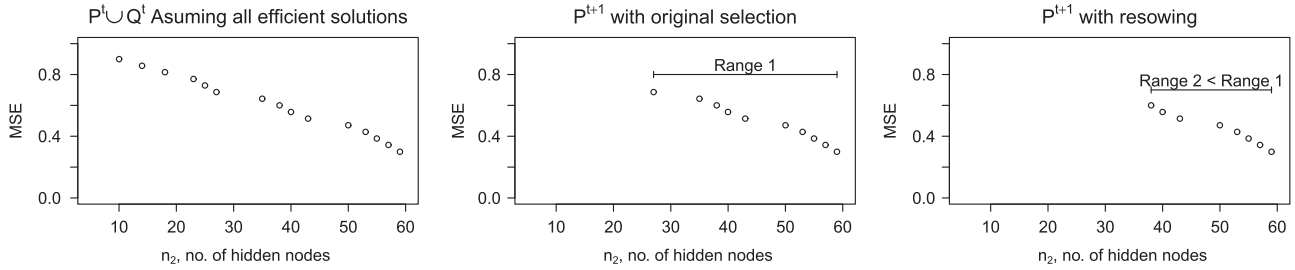

Fig. 2. MSE versus the number of hidden nodes of a set of 15 efficient solutions in $P^t \cup Q^t$ (left) and the solutions selected by the μG-ELM (center) and the μG2-ELM (right).

the other proposals, can be used in a great variety of meta-heuristic algorithms, both in their initialization stage and in the variation process. In [35] a random guided process is established for setting the hidden weights of an ELM algorithm to ensure the full column rank of matrix **H** (Section 2) and to increase the training and validation ability. An initialization with binary or ternary values for ELM is introduced in [37]. Finally, an empirical study is done in [38] which shows that the performance of SLFNs

Table 1
Data sets description. All data sets without reference have been obtained from [42].

Name	# of inputs	# of samples	Name	# of inputs	# of samples
Continuous/regression					
Ikeda(I) [43]	2	1000	Ikeda(R) [43]	2	1000
MGTS [44]	4	1500	Chi2 [45]	2	1000
Friedman [46]	5	1000	Gamma [45]	1	1000
SinC [11]	1	1000	Abalone	8	4177
Autompg	7	392	Autoprice	15	159
Bank	8	4499	Cleveland	14	297
Housing	13	506	Machine	7	209
Wisconsin	32	194			

Name	# Inputs/ outputs	# of samples	Name	# Inputs/ outputs	# of samples
Classification					
Australian card	14/2	690	Breast cancer	30/2	569
Heart	13/2	303	Ionosphere	33/2	351
Image	18/7	2310	Iris	4/3	150
Lymphography	18/4	148	Pima	8/2	768
Satellite Image	36/7	6435	Waveform	21/3	5000
Wine	13/3	178			

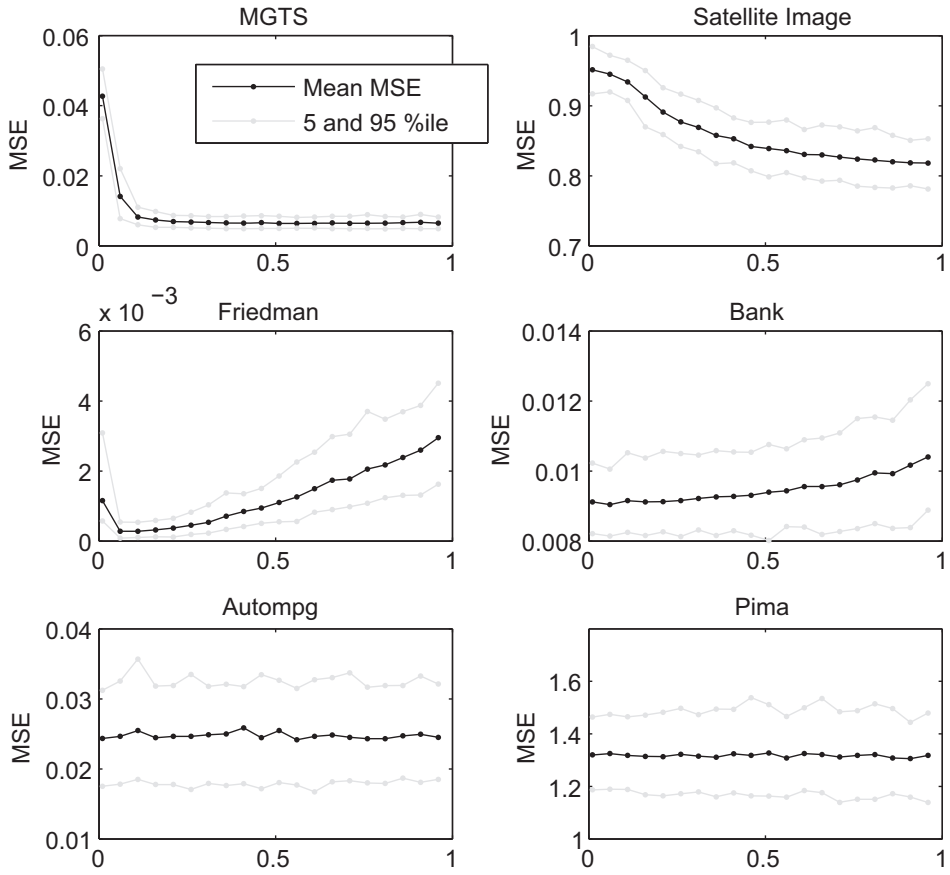

Fig. 3. Mean of the MSE obtained in the set of validation patterns and the percentiles 5 and 95 versus different values of σ_0 varying in the interval (0,1).

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

e_{ij}	$n_{2,1}$	$n_{2,2}$	$n_{2,3}$	$n_{2,4}$
σ_1	0.80	0.65	0.22	0.16
σ_2	0.90	0.36	0.88	0.74
σ_3	0.61	0.97	0.76	0.79
σ_4	0.10	0.88	0.23	0.94

Selected alternative	
Min min rule:	σ_4
Min max rule:	σ_1
Relative lost rule:	σ_3

Fig. 4. Results of the application of the three different decision rules to a hypothetical matrix of errors.

is usually better when input-to-hidden weights and bias are randomly assigned instead of fixed.

Our process establishes that the range of variation of the weights of the networks in the initial population depends on a parameter σ_0. After that, the own evolution of the algorithm will modify this value. Previous to present the device designed to obtain the appropriate value of the σ_0 parameter we show its advisability by means of a simple experiment. We have selected some representative data sets of the collection of problems that we will use later (see Table 1) and we have fixed several values of $\sigma_0 \in (0, 1)$. For each data set and each value of σ_0 one hundred of different initial populations are built. All the initial parameters $(\vec{a}, \vec{b}, n_2)$ are obtained from a normal distribution with mean equal to zero and standard deviation equal to σ_0. After that, the algorithm that we proposed is applied for each initial population. Fig. 3 shows the mean of the MSE obtained in the set of validation patterns and the percentiles 5 and 95 (see in Section 4 how the data sets are divided into training, test and validation sets).

Fig. 3 shows how initial values affect the MSE depending on the data set. In the first row the smallest values of MSE correspond to high values of σ_0, contrary to what happens in the second row. In the third row two examples for which σ_0 does not affect the value of the MSE are shown.

Therefore, we confirm the need of designing a device for selecting the appropriate initial values which is described in the following paragraphs.

The ideal way to proceed would be to look for the appropriate set of values of σ_0, one for each of the input weights associated with each random hidden neuron, and one more for its bias. But, in such a case we would have to propose a very complex experimental design for selecting them which will negatively affect the computational effort of our algorithm (we want a fast tool which does not much delay our algorithm). As a consequence, we suggest a fast and simple process, framed in the decision theory field [41], to select the initial value of σ_0 (the same for all the parameters) and, then, start the algorithm in better conditions than originally. It is worthy to notice that this new device can be used in any other metaheuristic algorithm which requires to initialize the weights and bias of a neural network.

For each data set, we design a decision problem in which we have to select one alternative (the value of σ_0) under several unpredictable scenarios (the appropriate number of hidden nodes which is not known in advance) in order to minimize a lost function (the corresponding MSE). So, we consider that we have s possible values of σ_0, $0 < \sigma_{0i} \leq 1, i = 1, \ldots, s$ (s alternatives) and t possible values of the number of hidden nodes, $0 < n_{2,j} \leq n_{max}, j = 1, \ldots, t$ (t scenarios). Then, we apply the training method for a SLFN and we consider as measure of goodness the MSE (e_{ij}, the error under the alternative σ_{0i} and the scenario $n_{2,j}$) evaluated in a certain set of patterns.

After this initial process, to obtain the preferred alternative we have to establish a decision rule. We have selected three rules from the different well-known decision rules [41, pp. 59–63], because of their simplicity and also because they collect three wide points of view: the optimistic, where σ_{0i} is chosen to obtain the minimum e_{ij} for all the scenarios; the pessimistic, which chooses the best one in the worst scenario; and, finally, the conservative, which chooses the value of σ_{0i} that minimizes the maximum difference when the best decision is not made. The three methods select the alternative σ_{0k} such that a certain error measure, $e_i, i = 1, \ldots, s$, is minimized

$$k = \arg \min_{1 \leq i \leq s} \{e_i\} \tag{6}$$

and the decision rules are the following:

- Minmin Criterion (Optimistic rule) where e_i is obtained as

$$e_i = \min_{1 \leq j \leq t} \{e_{ij}\}, \quad i = 1, \ldots, s \tag{7}$$

  and it represents the minimum error reachable if we consider the alternative σ_{0i}.
- Minmax Criterion (Pessimistic rule) where e_i is obtained by means of

$$e_i = \max_{1 \leq j \leq t} \{e_{ij}\}, \quad i = 1, \ldots, s \tag{8}$$

  and it represents the maximum error reachable if we consider the alternative σ_{0i}.
- Minmax regret Criterion (Relative Lost rule) which defines the matrix regret as

$$r_{ij} = e_{ij} - \min_{1 \leq k \leq s} \{e_{kj}\}, \quad i = 1, \ldots, s, \ j = 1, \ldots, t \tag{9}$$

  the difference between the corresponding error and the best alternative for that scenario, and e_i is obtained as

$$e_i = \max_{1 \leq j \leq t} \{r_{ij}\}, \quad i = 1, \ldots, s. \tag{10}$$

Fig. 4 shows the obtained results when we apply these decision rules to a hypothetical matrix of errors e_{ij} obtained with four scenarios (σ_{0i}, $i = 1, \ldots, 4$) and four alternatives ($n_{2,j}$, $j = 1 \ldots, 4$). The optimistic criterion selects the highest value of σ_0, the pesimistic the lowest and an intermediate value is selected by the conservative rule.

Our proposal is to include a device to be applied just before selecting the initial population. It is described in Procedure 4. In this device a small number k of SLFNs are generated for each value of σ_i, $i = 1, \ldots, s$, and each number of hidden nodes $n_{2,j}$, $j = 1, \ldots, t$. Then, Eq. (3) is applied with the training patterns to obtain the output weights and then, Eq. (4) to evaluate the networks in the test set. The best of the k SLFNs gives the value of the error measure. Finally, one of the three decision rules is applied. In Section 4 an experiment will be conducted in order to determine the rule or rules which get the best behavior for our algorithm.

Procedure 4. Determining the value of the initial parameter σ_0 (k=2).

Set s and t
Set alternatives σ_i, $i = 1, \ldots, s$
Set scenarios $n_{2,j}$, $j = 1, \ldots, t$
for $i \leftarrow 1$ **to** s **do**
  for $j \leftarrow 1$ **to** t **do**
    for $m \leftarrow 1$ **to** 2 **do**
      Generate a_{lk}^m, b_k^m from a $N(0, \sigma_i)$, $l = 1, \ldots, n_1, k = 1, \ldots, n_{2,j}$
      $\beta^m = \mathbf{H}^{\dagger,m}\mathbf{T}$ (Eq. (3), in the training set)
    end for
    $e_{ij} = \min(MSE(\vec{a}^1, \vec{b}^1), MSE(\vec{a}^2, \vec{b}^2))$ (Eq. (4) in the test set)
  end for
end for
Apply Eqs. (7) (or 8) or (9)–(10) and then, Eq. (6) to obtain the value of σ_0

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

4. Tuning our algorithm

Since the main goals of the modifications are both to increase the exploratory capacity of the algorithm and to start with the optimal initial conditions, we do not consider the study of the effect of each modification separately but all together as a whole.

In this section we develop the experiments carried out to tune our algorithm. In this tunning process we have to determine the strategy for selecting the individuals and for preprocessing them, before applying the regression device, [28, Sec. 4]. Furthermore, we have to choose one of the decision rules proposed in Section 3.4 to initialize the population.

The experiments are achieved with the data sets shown, together with their main characteristics, in Table 1. They are classified according to their type of underlying problem: to approximate a continuous function, regression and classification. All are standardized into the range $[-1, 1]$. These data sets will be also used in Section 5 to compare our algorithm with its competitors.

In our previous paper, [28, Sec. 4] we have proposed three strategies for selecting the individuals before applying the regression device: S_1, S_2 and S_3 (which depend on the set of efficient solutions we consider) and three ways to preprocess them: P_1, P_2 and P_3 (which depend on the treatment applied to extreme values in the solutions set). Now, we also consider three versions of our algorithm, one for each decision rule presented in Section 3.2: μG-ELM-O, μG-ELM-P and μG-ELM-RL, which correspond to Optimistic, Pessimistic and Relative Lost rules, respectively. For each version we have to decide which combination of the two factors S_i, $i=1,\ldots,3$, and P_j, $j=1,\ldots,3$, presents the best performance. After that, the three versions are compared to select the best for each kind of problem: continuous, regression and classification.

The development of the first part of this experiment is similar to that made in [28, Seq 51]. Each data set is used to train 100 times a neural network using each version of the algorithm and under each of the nine combinations of the two factors. Then, we obtain 100 final neural network models for each problem and for each configuration version-combination of the factors. For each of the 100 processes, the corresponding data set is randomly divided in three subsets: 56.66% for training, 10.00% for testing (both used in the learning process) and 33.33% for the validation set used for evaluating the final ability of generalization. The parameters that we used for running the algorithms are the same we used in the previous work: size of population and number of iterations equal to 25; maximum number of hidden nodes equal to 100, number of solutions to be evolved in the micro-loop equal to 4 (for a complete description of the parameters we refer the readers to [28]). In addition to the new parameters, the re-sowing percentage and the values of σ_i and n_{2j} are also fixed and they are shown in Table 2.

After this process we compute the MSE/CEP in the validation set and we record the final number of hidden nodes, both used to determine the best alternative. A Friedman test [47, pp. 272–284] has been applied for comparing the MSE/CEP and the number of hidden nodes of the resulting network for each group of problems: continuous, regression and classification.

If significant differences are found among the configurations then a multiple comparison Wilcoxon test [47, pp. 295–300] is carried out to make decisions about the individual differences between pairs of configurations. We fix the significance level equal to 0.05. For simplicity, the nine combinations S_i, P_j are numbered from 1 to 9.

Table 2
Additional parameter setting.

Parameter	Value
Re-sowing	10%
σ_i	0.05, 0.15, 0.35, 0.5, 0.75
n_{2j}	15, 40, 65, 90

Table 3
Best homogeneous means subsets for error (MSE/CEP) and number of hidden nodes (n_2). The combinations appear in increasing value of the rank provided by the Friedman test. Combinations with the same superscript can be considered equivalent.

μG-ELM-O	
Continuous	
MSE	3^a $\{6\ 2\ 5\ 9\ 1\}^{ab}$ $\{8\ 7\ 4\}^b$
n_2	$\{7\ 4\}^a$ $\{8\ 9\ 1\ 6\}^{ab}$ $\{5\ 2\ 3\}^b$
Regression	
MSE	$\{2\ 7\ 8\ 9\ 4\ 3\ 5\ 6\ 1\}^a$
n_2	4^a 6^{ab} $\{5\ 1\}^b$ $\{7\ 3\ 9\}^{bc}$ $\{2\ 8\}^c$
Classification	
CEP	$\{2\ 3\ 5\ 1\ 8\ 6\ 9\ 4\ 7\}^a$
n_2	4^a $\{6\ 5\}^{ab}$ $\{9\ 8\ 3\ 1\ 2\ 7\}^b$
μG-ELM-P	
Continuous	
MSE	$\{3\}^a$ $\{2\ 6\ 5\ 1\ 9\}^{ab}$ $\{8\ 7\ 4\}^b$
n_2	4^a $\{7\ 9\ 6\}^{ab}$ $\{8\ 1\ 5\ 2\ 3\}^b$
Regression	
MSE	$\{4\ 7\ 2\ 3\ 5\ 9\ 8\ 1\ 6\}^a$
n_2	$\{4\ 6\ 5\}^a$ $\{1\ 9\ 7\ 3\ 8\ 2\}^b$
Classification	
CEP	$\{5\ 3\ 2\ 1\ 7\ 6\ 9\ 8\ 4\}^a$
n_2	4^a $\{6\ 5\}^{ab}$ $\{9\ 3\ 2\ 8\ 1\ 7\}^b$
μG-ELM-RL	
Continuous	
MSE	3^a $\{6\ 2\ 5\ 1\ 9\}^{ab}$ $\{8\ 4\ 7\}^b$
n_2	4^a $\{7\ 8\ 9\}^{ab}$ $\{6\ 1\ 5\ 2\ 3\}^b$
Regression	
MSE	$\{2\ 4\ 7\ 1\ 5\ 9\ 3\ 8\ 6\}^a$
n_2	$\{4\ 6\ 5\}^a$ $\{1\ 7\ 3\ 9\ 2\ 8\}^b$
Classification	
CEP	$\{2\ 5\ 1\ 3\ 6\ 7\ 9\ 8\ 4\}^a$
n_2	4^a $\{5\ 6\}^{ab}$ $\{9\ 7\ 1\ 3\ 2\ 8\}^b$

In Table 3 we can observe significant differences in the MSE/CEP for each of the three algorithms only in continuous problems and for n_2 for each algorithm and kind of problem. So, for selecting the best option for regression and classification problems we take into consideration the number of hidden nodes. This leads to select combination fourth, that is, S_2P_1, for the three algorithms, which is the best choice for n_2 in all the situations. For the continuous case the best combinations for MSE are the worst for n_2 and vice versa. So we have to take a decision that prioritizes some element. We decide to prioritize the MSE. Then, we chose combination sixth (S_2P_3), which is in the first subset for MSE in the three versions of the algorithm and it is in an intermediate position for n_2 (second subset, superscript b). We want to point out that in order to simplify the process, we have selected one combination common for the three algorithms for each kind of problem.

The second part of the experiment consists of choosing one of the three algorithms for each type of problem. We carry out an analogous experiment using the results obtained with the corresponding combination S_iP_j chosen previously for each type of problem. In this case the Friedman test only finds significant differences among the algorithms for the MSE in continuous problems where μG-ELM-O and μG-ELM-RL show better behavior than μG-ELM-P. For n_2 the three algorithms are equivalent. Then, we select μG-ELM-RL, the one using the conservative strategy, as the best version of the algorithm since it is the one with the smallest mean rank for n_2, even though they are statistically equivalent. Then, it will be the version used in Section 5 to be compared with the competitors. We call it μG2-ELM.

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

5. Comparison of the algorithms

Now, we develop the experiment carried out to study the behavior of our algorithm compared with the behavior of other algorithms in the literature. Unlike the previous experiment in Section 4 now we study the results in a more detailed way, considering problem by problem instead of the three groups of problems.

The new algorithm μG2-ELM is compared with the original μG-ELM, with a classical Back Propagation algorithm implemented in the Neural Network Tool Box of Matlab software [48] and with a set of validated ELM algorithms: the original ELM [11] and the online sequential ELM, OS-ELM [17], which allows training data to come one by one or chunk by chunk; the ensemble of online sequential ELM, EOS-ELM [18], which combines neural network ensembles and OS-ELM; the enhanced I-ELM, EI-ELM [13] (an improved version of the incremental ELM, I-ELM [19]) in which at each step a certain number of hidden nodes are built and only the one reaching the biggest training error reduction is selected; the optimally pruned ELM, OP-ELM [20], which builds a network using ELM and then removes no significant hidden neurons by using regression techniques; SaE-ELM [23], which is an evolutionary algorithm based on ELM and differential evolution; and, finally, the weighted ELM, W-ELM [24], an ELM for imbalanced data which re-weight the input data by adding different penalty coefficients to the training errors corresponding to different inputs. This algorithm is considered in such a way that for continuous and regression problems the proposed weighting matrix W is set to the identity. Next two methods have been mentioned in Section 1, but they are not finally considered since they lead to not promising results. The error minimized ELM, EM-ELM [21] is an error minimization based method in which the number of hidden nodes can grow one-by-one or group-by-group until optimal and the TS-ELM [22] is a two-stage algorithm which, in the first stage, groups of hidden nodes are added to the network until the stopping criterion is met and then, in the second stage the hidden nodes are reviewed and the insignificant ones are removed.

For each algorithm except for EI-ELM, OS-ELM and EOS-ELM, we have used the codes and the parameters provided by their authors. The other three algorithms have been implemented using the descriptions and parameters presented in the corresponding papers.

Among the selected algorithms only our algorithms, OP-ELM and EI-ELM give as solution to the appropriate number of hidden nodes as well as the appropriate weights associated with this number. The remaining algorithms look for the weights of a neural network in which the number of hidden nodes has been set previously. However, we also include in this second group the EI-ELM since it needs to set an upper bound for the error to be reached and this value depends on the particular problem being solved. We have decided to set this bound to 0, then, the EI-ELM will never reach it and it will stop when the maximum number of hidden nodes is reached. Therefore, we have designed two kind of executions. The algorithms in the first group (μG-ELM, μG2-ELM and OP-ELM) are executed and the neural network obtained is evaluated using the validation set in order to obtain the MSE/CEP. For the remaining algorithms, one execution of the algorithm is the application of the algorithm with $1, 2, \ldots, n_{max}$ hidden neurons. Then, the MSE/CEP evaluated in the test set is obtained for each network and the one with the best MSE/CEP is selected. Next, this neural network is evaluated in the validation set to obtain the corresponding MSE/CEP.

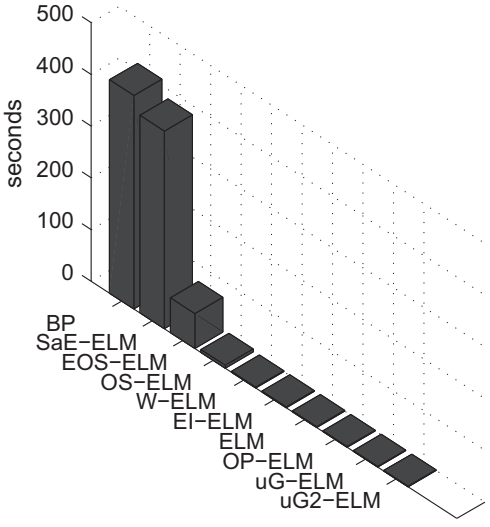

Fig. 5. Mean time of execution per problem in seconds sorted from the highest to the lowest.

Furthermore, in order to make the comparison as fair as possible, we fix a minimum common time for the execution of all the algorithms for each data set. This time is obtained as the mean time of execution of our algorithm, μG2-ELM, for each of the data sets. So, for each data set, each algorithm carries out a sequence of complete executions until the accumulated time of execution is greater than the reference time. The result is that all the competitors present longer time of execution than ours, as it can be observed in Fig. 5.

For comparing our algorithm with each of its competitors a pairwise mean test for independent samples with a significance level of 5% has been applied.

Tables 4–6 show the mean values obtained for the MSE/CEP and the number of hidden nodes by all the algorithms in all the data sets. The numbers in the columns of the competitors, including as a competitor of our original version, are in normal, italic or boldface style. Normal style indicates that significant differences have not been found between the algorithms, italic means that there are significant differences and the competitor performance is better, and boldface means that significant differences are detected and our performance is better.

First, we will compare the behavior of μG2-ELM versus the previous version μG-ELM (column 3 in Tables 4–6). After that, we will compare our new algorithm with the remaining ones (columns 4–11 in Tables 4–6).

When we compare the new version of our algorithm, μG2-ELM, versus the previous version μG-ELM, we can see that the new version reaches better mean for MSE/CEP in 5 out of 26 problems and they are statistically equivalent in the remaining ones except for MGTS, Chi2 and Satimage. Regarding the number of hidden nodes, the new version gets better results in 6 problems. In four of them, Ikeda(I), Ikeda(R), Friedman1 and Bank, the reduction is in both the mean value of MSE and the number of neurons. In Chi2 and Satimage the lower number of hidden nodes is at the expense of the error. The new version is worst only in 1 problem, Image, for which the increase in the number of hidden nodes provides a better CEP.

Therefore the μG2-ELM exhibits a slightly better performance than its ancestor. For continuous problems the new proposal is better in both the mean error and the number of hidden nodes, in 3 out of 7 problems and, regarding the number of neurones, it is better in one more and it is never worse than the original version. For regression problems, it improves both the mean error and the number of hidden nodes in 1 out of 8 and its performance is equal for the rest. Finally, for classification problems both are similar since they present equal results in all the problems except for Image for which the performance of μG2-ELM is better in the number of nodes by worsening the mean error and Satimage for which the contrary occurs.

It is obvious that we do not obtain a dramatic improvement since the previous version of the algorithm got values of the MSE and number of nodes much better than most of the rest of

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

Table 4
Mean values of the MSE and the number of hidden nodes, n_2, for the continuous problems evaluated in the validation set.

Data set	μG2-ELM	μG-ELM	ELM	W-ELM	SaE-ELM	EI-ELM	OP-ELM	OS-ELM	EOS-ELM	BP
Continuous										
Ikeda(I)										
MSE	4e−16	**1e−12**	*2e−16*	**1e−09**	*2e−16*	**0.214**	**6e−08**	**7e−14**	3e−08	1e−07
n_2	10.00	**22.16**	**42.97**	**84.36**	**43.20**	**97.01**	**61.35**	**46.18**	**78.47**	**18.43**
Ikeda(R)										
MSE	4e−16	**2e−12**	*1e−16*	**1e−09**	*2e−16*	**0.208**	**7e−08**	**5e−14**	**2e−10**	**3e−08**
n_2	10.00	**23.05**	**43.22**	**81.75**	**43.37**	**96.84**	**60.55**	**44.43**	**83.36**	**15.49**
MGTS										
MSE	7e−03	*6e−03*	**9e−03**	**0.013**	**9e−03**	**0.172**	**0.011**	**0.022**	**0.013**	*3e−03*
n_2	98.34	98.91	*97.39*	*95.75*	*96.38*	98.39	*91.40*	*54.93*	*93.77*	*82.82*
Chi2										
MSE	2e−03	*1e−03*	**0.016**	**0.026**	**8e−03**	**0.529**	2e−03	**0.041**	**0.031**	*8e−04*
n_2	67.00	**81.17**	**92.48**	**80.79**	**90.84**	**98.38**	**80.60**	*15.47*	**79.54**	**73.58**
Friedman										
MSE	3e−04	**1e−03**	**9e−04**	**8e−04**	**1e−03**	**0.056**	**0.015**	**1e−03**	**4e−03**	*2e−05*
n_2	96.64	**98.44**	*95.11*	*95.16*	*94.06*	97.62	*66.40*	*93.70*	*79.34*	*40.50*
Gamma										
MSE	2e−05	2e−05	2e−05	**1e−04**	2e−05	**0.376**	*1e−05*	**3e−05**	**1e−04**	*8e−06*
n_2	13.18	13.84	**38.34**	**89.65**	**37.24**	**98.28**	**18.00**	**65.82**	**68.66**	**40.44**
SinC										
MSE	1e−08	2e−08	**1e−06**	**0.174**	**1e−06**	**0.349**	*8e−11*	**2e−06**	**0.171**	7e−09
n_2	17.07	17.57	**69.10**	**60.56**	**70.16**	**69.21**	**32.60**	**77.57**	**41.53**	**37.84**

Table 5
Mean values of the MSE and the number of hidden nodes, n_2, for the regression problems evaluated in the validation set.

Data set	μG2-ELM	μG-ELM	ELM	W-ELM	SaE-ELM	EI-ELM	OP-ELM	OS-ELM	EOS-ELM	BP
Regression										
Abalone										
MSE	0.024	0.024	0.025	0.025	**0.026**	**0.038**	3967	0.024	**0.026**	*0.023*
n_2	31.43	35.00	**60.47**	**61.85**	**59.68**	**96.12**	**46.15**	**52.04**	**65.31**	*21.11*
Autompg										
MSE	0.024	0.024	**0.027**	**0.028**	**0.027**	**0.052**	0.046	**0.027**	**0.028**	**0.028**
n_2	25.01	22.94	**42.74**	**45.75**	**45.04**	**92.09**	**34.15**	**41.67**	**57.72**	**36.68**
Autoprice										
MSE	0.041	0.043	**0.064**	**0.057**	**0.057**	0.044	12.2	**0.061**	**0.090**	**0.052**
n_2	15.20	15.28	**31.01**	**33.63**	**28.42**	**69.62**	**19.55**	**29.91**	**43.51**	**37.06**
Bank										
MSE	8e−03	**9e−03**	**9e−03**	**9e−03**	**9e−03**	**0.083**	**0.038**	**9e−03**	**1e−02**	*6e−03*
n_2	28.73	**60.84**	**89.35**	**90.54**	**88.72**	**98.91**	**77.95**	**89.50**	**79.19**	*12.00*
Cleveland										
MSE	0.207	0.208	**0.230**	**0.228**	**0.229**	0.205	**0.230**	**0.227**	**0.236**	**0.252**
n_2	12.15	10.95	**27.67**	**30.21**	**27.82**	**73.45**	**20.15**	**28.13**	**46.44**	**22.46**
Housing										
MSE	0.035	0.038	0.039	**0.039**	0.038	**0.078**	78.3	0.039	**0.041**	0.035
n_2	40.34	45.67	**74.55**	**74.92**	**71.85**	**93.47**	36.60	**72.98**	**75.96**	*34.20*
Machine										
MSE	2e−03	3e−03	**7e−03**	4e−03	5e−03	**0.017**	0.929	**6e−03**	0.020	**7e−03**
n_2	24.00	22.34	**43.85**	**67.16**	**41.07**	**76.85**	*15.35*	**35.94**	**60.01**	*13.62*
Wisconsin										
MSE	0.743	0.740	**0.865**	**0.882**	**0.865**	0.696	0.797	0.814	**1.173**	**0.845**
n_2	11.83	10.68	**27.92**	**29.77**	**26.37**	**64.42**	14.00	**25.31**	**50.90**	**39.49**

versions. Therefore to obtain these slight but statistically significant improvements is a big deal.

Now, we compare our μG2-ELM with its competitors. First, we center the analysis in BP (the last column in Tables 4–6) which is the only one not ELM based. Also it is the only one which can be considered equivalent to the μG2-ELM in terms of the error and the number of nodes. Regarding the error we obtain better results in 2 continuous problems (Ikeda(I) presents a worse error than ours but statistically significant differences have not been detected due to the huge variability of its results), 5 regression problems and 3 classification problems, but BP beats us in 4 continuous problems, 2 regression problems and 3 classification problems. When looking at the number of nodes, we are better in 5 continuous problems, 4 regression problems and 3 classification problems, but BP gets lower number of neurons in 2 continuous problems, 4 regression problems and 6 classification problems.

Therefore, BP performance is very similar to our μG2-ELM, but, BP is not competitive regarding the computational point of view, since its execution time is from 30 to 1100 times greater than ours. Fig. 5 shows that the mean time of execution of BP is the highest one.

When attending to the algorithms based on the ELM methodology, our algorithm reaches a lower error in 39 out 49 entries, 79.6%, for continuous problems (Table 4), 37 out of 56 for regression problems, 66.1%, (Table 5) and 51 out of 77 for classification problems, 66.2%, (Table 6). And we are only worse in 6, 0 and 3 (12.3%, 0% and 3.9%), respectively. It is worth noting that in regression problems, the pairwise test cannot detect significant differences despite the high values obtained with OP-ELM in problems Abalone, Autompg, Autoprice, Housing and Machine due to the huge variability of the obtained results.

In addition, we are always better than EI-ELM, OS-ELM, EOS-ELM and W-ELM for all continuous problems, except for Ikeda(I) for which

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

Table 6
Mean values of the CEP and the number of hidden nodes, n_2, for classification problems evaluated in the validation set.

Data set	μG2-ELM	μG-ELM	ELM	W-ELM	SaE-ELM	EI-ELM	OP-ELM	OS-ELM	EOS-ELM	BP
Classification										
Aus.Card.										
CEP	14.04	14.33	14.23	**14.64**	14.47	14.20	**14.65**	14.30	14.46	**14.67**
n_2	19.99	19.29	**45.81**	**48.45**	**44.77**	**80.76**	20.65	**44.62**	**60.10**	17.05
Bcancer										
CEP	03.66	03.81	**04.38**	**04.10**	**04.31**	**07.15**	**05.34**	**04.40**	**04.66**	03.56
n_2	37.48	34.32	**63.60**	**66.34**	**52.95**	**90.57**	*19.95*	**62.13**	**77.72**	*19.95*
Heart										
CEP	18.36	18.38	**20.11**	**21.66**	**20.63**	17.78	**19.45**	**20.00**	**20.43**	**20.49**
n_2	11.82	13.45	**32.43**	**35.57**	**44.08**	**75.89**	**16.85**	**28.86**	**51.53**	**18.26**
Ionosph.										
CEP	12.25	11.70	12.74	**14.27**	11.54	12.87	*10.03*	12.62	**13.43**	12.03
n_2	22.31	22.18	**42.41**	**48.11**	**48.37**	**89.01**	**36.55**	**43.17**	**67.12**	25.76
Image										
CEP	06.19	**07.02**	**06.54**	**06.52**	*05.50*	**21.00**	**09.36**	**06.64**	**06.97**	*03.84*
n_2	80.96	*75.93*	**93.60**	**94.01**	**94.97**	**99.87**	77.25	**91.34**	**88.00**	*37.28*
Iris										
CEP	04.50	04.52	**05.90**	03.94	**06.18**	**17.75**	04.75	**05.60**	**05.83**	04.90
n_2	18.84	17.65	**27.72**	**45.58**	**29.32**	**83.37**	17.25	**24.85**	**44.78**	*13.28*
Lymphog.										
CEP	22.73	22.57	23.88	**24.88**	**24.73**	*20.24*	23.51	23.49	**25.86**	23.84
n_2	16.17	15.74	**26.27**	**35.09**	**29.47**	**70.85**	**23.30**	**26.46**	**49.21**	**20.59**
Pima										
CEP	23.34	23.79	23.86	**28.71**	**24.22**	**27.30**	**24.49**	23.82	23.70	**24.06**
n_2	13.79	12.59	**33.15**	**44.98**	**38.73**	**94.39**	14.25	**33.70**	**48.59**	**27.44**
Satimage										
CEP	12.33	*12.12*	**13.07**	**15.39**	13.23	**24.61**	**13.12**	**13.11**	**13.52**	*10.14*
n_2	92.92	**98.37**	**96.08**	95.26	92.42	**99.92**	**98.85**	**95.79**	*88.45*	*78.76*
Wavef.										
CEP	14.26	14.45	**14.56**	**15.16**	**14.61**	**17.50**	**16.00**	**14.49**	**14.84**	*13.91*
n_2	95.52	96.11	*91.52*	*91.68*	*88.84*	**99.27**	*77.05*	*90.69*	*85.91*	*14.35*
Wine										
CEP	03.74	03.45	04.44	**04.90**	03.85	03.97	**05.36**	04.15	**06.43**	04.47
n_2	20.31	22.33	**36.48**	**42.48**	**37.47**	**96.26**	**31.30**	**35.27**	**69.11**	*10.77*

significant differences are not detected for EOS-ELM, but it is due to the huge variability of its results. In regression problems, we are better or equal than all the considered methods. OP-ELM is now the technique with more variability and significant differences could not be detected in Abalone, Housing, Machine and Autoprice even though their errors are much higher than ours. Finally, we are better or equal than ELM, W-ELM, OS-ELM and EOS-ELM for classification problems and SaE-ELM, EI-ELM and OP-ELM only win us once in Image, Lymphography and Ionosphere, respectively.

Regarding the number of hidden nodes, we obtain a lower number in 34 out of 49 entries for continuous problems, 69.4% (Table 4), in 53 out of 56 for regression problems, 94.6% (Table 5) and 63 out of 77 for classification problems, 81.8% (Table 6). Furthermore, we are only worse in 13, 1 and 8 (26.5%, 1.8% and 10.4%), respectively.

In particular, we are better or equal than EI-ELM and OP-ELM, which are those which try to improve the compactness of the ELM, in 12 out of 14 (85.7%) for continuous problems, 15 out of 16 (93.8%) for regression problems and 20 out of 22 (90.9%), for classification problems. It is again worth noting that we are always better than EI-ELM for all regression and classification problems and better or equal in continuous problems. Furthermore, we have a very good behavior in regression problems where only OP-ELM is better than us for Machine. Furthermore, despite we are worse in number of nodes than almost all the algorithms for MGTS, Friedman1 and Waveform, our MSE/CEP is better.

We also want to point out that we perform much better than W-ELM for both balanced and unbalanced data sets in classification problems. Also, our behavior is better than SaE-ELM, the evolutionary algorithm that we have considered. μG2-ELM overcomes it in MSE and number of nodes in all the regression problems except in two for which we perform equal. In continuous problems we obtain better or equal MSE in 5 out of 7 problems and better or equal number of nodes also in 5 (85.7%). In classification problems we are better or equal in all the problems (11) except in Waveform for which SaE-ELM obtains a smaller number of nodes and Image for which SaE-ELM gets a lower CEP.

We conclude that our μG2-ELM is very competitive when compared with other ELM algorithms, in particular, when attending to the complexity of the resultant neural network. Only BP beats us in this respect but it is computationally much more expensive.

6. Conclusions

In this paper we have proposed a bi-objective genetic algorithm based on the ELM methodology, μG2-ELM, which improves the performance of that presented by the authors in [28]. The improvements include the use of a Gaussian distribution to set and mutate the weights and bias of the hidden nodes, the design of a decision problem to establish the initial values of the hidden weights and bias of the elements of the initial population and a resowing process to select the set of solutions which are used in each new iteration of the algorithm.

The algorithm has been compared with its previous version, a set of ELM algorithms and one back-propagation algorithm showing a very good behavior when the error reached and the number of hidden neurons are considered. As final conclusion, the authors consider that the synergy between the generation of solutions by using evolutionary algorithms and the evaluation by using ELM methodology increases the learning performance of both approaches compared with that obtained when they are applied separately. Therefore, it is worthy to take advantage of this synergy and to continue with the development of algorithms which combine both methodologies.

Please cite this article as: B. Lacruz, et al., μG2-ELM: An upgraded implementation of μ G-ELM, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.07.069

Acknowledgment

This research has been supported by research groups of Gobierno de Aragón E58 and E22.

References

[1] Rezaul K. Begg, R. Sarker (Eds.), Neural Networks in Healthcare: Potential and Challenges, IGI Global, 2006.
[2] M. Yasmin, M. Sharif, S. Mohsin, Neural networks in medical imaging applications: a survey, World Appl. Sci. J. 22 (1) (2013) 85–96.
[3] J. Ramírez-Quintana, M. Chacon-Murguia, J. Chacon-Hinojos, Artificial neural image processing applications: a survey, Eng. Lett. 20 (1) (2012) 1–13.
[4] G. Lalithamma, P. Puttaswamy, Literature review of applications of neural network in control systems, Int. J. Sci. Res. Publ. 3 (9) (2013) 1–6.
[5] F. Stahl, I. Jordanov, An overview of the use of neural networks for data mining tasks, WIREs Data Mining Knowl. Discov. 2 (2012) 193–208. http://dx.doi.org/10.1002/widm.1052.
[6] I. Flood, Neural networks in civil engineering: a review, in: B. Topping (Ed.), Civil and Structural Engineering Computing, Saxe-Coburg Publications, 2001, pp. 185–209.
[7] K. Smith, J. Gupta, Neural networks in business: techniques and applications for the operations researcher, Comput. Oper. Res. 27 (2000) 1023–1044.
[8] R. Dase, D. Pawar, Application of artificial neural network for stock market predictions: a review of literature, Int. J. Mach. Intell. 2 (2) (2010) 14–17.
[9] D. Baptista, F. Morgado-Dias, A survey of artificial neural network training tools, Neural Comput. Appl. 23 (3–4) (2013) 609–615.
[10] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: 2004 IEEE International Joint Conference on Neural Networks, vol. 70, 2004, pp. 25–29.
[11] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (13) (2006) 489–501.
[12] G. Huang, G.-B. Huang, S. Song, K. You, Trends in extreme learning machines: a review, Neural Netw. 61 (2015) 32–48.
[13] G. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16–18) (2008) 3460–3468.
[14] G. Huang, D. Wang, Y. Lan, Extreme learning machines: a survey, Int. J. Mach. Learn. 2 (2011) 107–122.
[15] E. Cambria, G. Huang, L. Kasun, E. Al, Extreme learning machines [trends & controversies], IEEE Intell. Syst. 28 (6) (2013) 30–59.
[16] S. Ding, X. Xu, R. Nie, Extreme learning machine and its applications, Neural Comput. Appl. 25 (3–4) (2014) 549–556.
[17] N. Liang, G. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, IEEE Trans. Neural Netw. 17 (6) (2006) 1411–1423.
[18] Y. Lan, Y. Soh, G. Huang, Ensemble of online sequential extreme learning machine, Neurocomputing 72 (13–15) (2009) 3391–3395.
[19] G. Huang, L. Chen, C. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[20] Y. Miche, P. Bas, C. Jutten, O. Simula, A. Lendasse, A methodology for building regression models using extreme learning machine: OP-ELM, in: European Symposium on Artificial Neural Networks, Bruges, Belgium, 2008, pp. 1–6.
[21] G. Feng, G. Huang, Q. Lin, R. Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, IEEE Trans. Neural Netw. 20 (8) (2009) 1352–1357.
[22] Y. Lan, Y. Soh, G. Huang, Two-stage extreme learning machine for regression, Neurocomputing 73 (16–18) (2010) 3028–3038.
[23] J. Cao, Z. Lin, G.-B. Huang, Self-adaptive evolutionary extreme learning machine, Neural Process. Lett. 36 (2012) 285–305.
[24] W. Zong, G.-B. Huang, Y. Chen, Weighted extreme learning machine for imbalance learning, Neurocomputing 101 (2013) 229–242.
[25] A. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer, Berlin, 2007.
[26] C. Coello, G. Lamont, D. van Veldhuizen, Evolutionary Algorithms for Solving Multi-objective Problems, 2nd Edition, Springer, Berlin, 2007.
[27] C. Coello, List of references on evolutionary multiobjective optimization, 2014. ⟨http://www.lania.mx/ccoello/EMOO/EMOObib.html⟩.
[28] D. Lahoz, B. Lacruz, P. Mateo, A multi-objective micro genetic ELM algorithm, Neurocomputing 111 (2013) 90–103.
[29] D. Nguyen, B. Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, in: Proceedings of the International Joint Conference on Neural Networks IJCNN, SanDiego, CA, USA, 1990, pp. 21–26.
[30] P.L. Bartlett, The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, IEEE Trans. Inf. Theory 44 (2) (1998) 525–536.
[31] Y. Oussar, G. Dreyfus, Initialization by selection for wavelet network training, Neurocomputing 34 (1–4) (2000) 131–143.
[32] M. Fernández-Redondo, C. Hernández-Espinosa, Weight initialization methods for multilayer feedforward, in: Proceedings of the European Symposium on Artificial Neural Networks, Bruges, Belgium, 2001, pp. 119–124.
[33] E. Castillo, B. Guijarro-Berdiñas, O. Fontenla-Romero, A. Alonso-Betanzos, A very fast learning method for neural networks based on sensitivity analysis, J. Mach. Learn. Res. 7 (2006) 1159–1182.
[34] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition versus randomness in soft computing techniques, Appl. Soft Comput. 8 (2008) 906–918.
[35] Y. Wang, F. Cao, Y. Yuan, A study on effectiveness of extreme learning machine, Neurocomputing 74 (16) (2011) 2483–2490.
[36] L.D. Tavares, R.R. Saldanha, D.A.G. Vieira, Extreme learning machine with initialized hidden weight, in: Proceedings of the 12th IEEE International Conference on Industrial Informatics (INDIN), Porto Alegre, RS, Brazil, 2014, pp. 43–47.
[37] M. van Heeswijk, Y. Miche, Binary/ternary extreme learning machines, Neurocomputing 149 (2015) 187–197.
[38] R. Wang, S. Kwong, X.-Z. Wang, A study on random weights between input and hidden layers, Soft Comput. 16 (2012) 1465–1475.
[39] K. Deb, M. Goyal, A combined genetic adaptive search (GeneAS) for engineering design, Comput. Sci. Inf. 26 (1996) 30–45.
[40] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, Reading, Massachusetts, 1989.
[41] S. Hansson, Decision theory. A brief introduction, (KTH), Royal Institute of Technology, Stockholm, 2005.
[42] A. Asuncion, D. Newman, UCI Machine Learning Repository, 2014. http://archive.ics.uci.edu/ml.
[43] S. Hammel, C. Jones, J. Moloney, Global dynamical behavior of the optical field in a ring cavity, J. Am. Opt. Soc. 2 (4) (1985) 552–564.
[44] M. Mackey, I. Glass, Oscillation and chaos in physiological control system, Science 197 (1977) 289–297.
[45] V. Dhar, A. Tickoo, R. Koul, R. Dubey, Comparative performance of some popular artificial neural network algorithms on benchmark and function approximation problems, PRAMANA J. Phys. 74 (2) (2010) 307–324.
[46] J. Friedman, Stochastic gradient boosting, Technical Report 4, Department of Statistics, Standford University, 1999.
[47] M. Hollander, A. Douglas, Nonparametric statistical methods, 2nd Edition, John Wiley & Sons, New York, NY, USA, 1999.
[48] M. Beale, M. Hagan, H. Demuth, Neural Network Toolbox User's Guide, The MathWorks, Inc, Natick, MA, 2014.

Beatriz Lacruz is an associate professor of statistics in the University of Zaragoza, Spain. She obtained her Ph. D. degree in mathematics from Zaragoza University in 1998. Her research interests include the development of statistical techniques to data mining, pattern recognition and machine learning.

David Lahoz received the B.Sc. degree in mathematics from the University of Zaragoza, Spain. He is a Ph.D. student and an assistant professor in the School of Engineering and Architecture of the University of Zaragoza. His research interests include artificial neural networks and multi-objective evolutionary algorithms.

Pedro M. Mateo received its Ph.D. degree in Sciences from the University of Zaragoza, Spain, in 1995. Since 1998 he is an associate professor in the Department of Statistical Methods of the University of Zaragoza, Spain. His current research interests include machine learning, multi-objective evolutionary algorithms and simulation modeling.