Neurocomputing 73 (2010) 1438–1450

Contents lists available at ScienceDirect

Neurocomputing

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

A multi-objective memetic and hybrid methodology for optimizing the parameters and performance of artificial neural networks

Leandro M. Almeida *, Teresa B. Ludermir

Centre of Informatics, Federal University of Pernambuco, Av. Luiz Freire s/n, Cidade Universitária, 50732-970 Recife-PE, Brazil

ARTICLE INFO

Article history:
Received 19 March 2009
Received in revised form
1 October 2009
Accepted 25 November 2009
Communicated by A. Abraham
Available online 16 December 2009

Keywords:
Artificial neural networks
Particle swarm optimization
Evolutionary algorithms
Memetic algorithms and hybrid intelligent systems

ABSTRACT

The use of artificial neural networks implies considerable time spent choosing a set of parameters that contribute toward improving the final performance. Initial weights, the amount of hidden nodes and layers, training algorithm rates and transfer functions are normally selected through a manual process of trial-and-error that often fails to find the best possible set of neural network parameters for a specific problem. This paper proposes an automatic search methodology for the optimization of the parameters and performance of neural networks relying on use of Evolution Strategies, Particle Swarm Optimization and concepts from Genetic Algorithms corresponding to the hybrid and global search module. There is also a module that refers to local searches, including the well-known Multilayer Perceptrons, Back-propagation and the Levenberg–Marquardt training algorithms. The methodology proposed here performs the search using the aforementioned parameters in an attempt to optimize the networks and performance. Experiments were performed and the results proved the proposed method to be better than trial-and-error and other methods found in the literature.

1. Introduction

The power of Artificial Neural Networks (ANNs) has been demonstrated over the years by their successful use in many types of problems with different degrees of complexity and in different fields of application [16,8,41,17,7]. However, the use of an ANN implies another problem beyond that in which the neural network will be employed. This secondary problem regards the choice of model and appropriate set of parameters that, together, can effectively and efficiently solve the main problem. The choice of ANN model is easier than choosing its parameters, as the conception a ANN models is linked to a specific type of problem, whereas the parameters are variant continuous values within the work context of the problem [29,7,32]. The choice of the set of parameters involves difficulties such as the exponential number of parameters that need to be adjusted; the need for a priori knowledge on the problem domain and ANN functioning in order to define these parameters; and the presence of an expert when such knowledge is lacking [2,4,3].

In most cases, the choice of parameters is manually performed through of a trial-and-error method, which is a tedious, less productive and error-prone task. Furthermore, when the complexity of the problem domain increases and near-optimal networks are desired, manual searching becomes more difficult and unmanageable [2,4,3]. An optimal neural network is an ANN tailored to a specific problem, thereby having a smaller architecture, with faster convergence and a better generalization performance [4]. A near-optimal ANN is a neural network with specific, appropriate parameters chosen for a particular problem, with a better structure and final performance than an ANN found through trial-and-error [1,4].

Thus, the automatic searching for or fine tuning of ANN parameters has become the subject of many papers found in the literature. Methods devoted to automatically searching/tuning/optimizing ANN structures and parameters can be classified into those that use some kind of Evolutionary Algorithm (EA) and those that use a non-evolutionary or numeric approach, focused mainly on the manipulation of ANN architectures and weights [21]. However, transfer functions, training algorithms and the rates of these algorithms also exercise an influence over the final performance of the neural network [4,3]. An evolutionary search can involve the search for either some or all ANN parameters, provided that such information is correctly specified for the adopted search methodology [19,15].

Recently, there has been a perceptible increase in the use of EAs for the optimization of problem solving, including the automated design of ANNs. EAs are heuristic, stochastic methods based on populations made up of individuals with specific behavior similar to biological phenomena; they are robust and efficient at exploring an entire solution space of optimization problems [50,19,15]. Genetic Algorithms (GA) [26] are the main

* Corresponding author.
E-mail addresses: lma3@cin.ufpe.br, leandrolma@gmail.com (L.M. Almeida), tbl@cin.ufpe.br (T.B. Ludermir).

0925-2312/$ - see front matter Crown Copyright © 2009 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2009.11.007

kind of EA used to search for ANN parameters, but other methods, such as Evolution Strategies (ES) [6,19], Particle Swarm Optimization (PSO) [33], and Ant Colony Optimization (ACO) [18], have also been employed in this difficult task [31,39,38,24,51,50,4,23].

This paper presents a method for optimizing the parameters and performance of ANNs through a search for weights, architectures (nodes and layers), transfer functions and training algorithm rates. The proposed method is based on fully connected supervised Multi-layer Perceptrons composed of a combination of ES, PSO and GA as well as the Back-propagation (BP) [47,46] and Levenberg-Marquardt (LM) [36,40] training algorithms [29]. The main difference between the method proposed and others found in the literature is the combination of strong points from different methods, which are normally used separately in the same task. The result is a hybrid system [27,42] with memetic [45], multi-objective [28,15] behavior employed to optimize ANNs.

More specifically, the simplest mutation and self-adaptation mechanisms are taken from ES, to which PSO is incorporated, with its ability to explore a small portion of the entire search space and refine the solutions found by the ES. The solutions are submitted to a training phase in which the algorithms (BP and/or LM) are employed to execute a more refined/precise local search than that executed by PSO. The solutions are then submitted to a selection survival process (existing in GA). Thus, solutions are generated randomly to form the offspring and the process is repeated. Further details are described in the Section 4. The proposed method is a hybrid intelligent system, combining the strong points of the different techniques in order to overcome the limitations of each technique used alone [27,42]. This method has a stage in which the global search is performed using ES and PSO, followed by a stage in which the local search is performed using the ANN training algorithms and PSO throughout the execution, thereby characterizing memetic behavior [45,19]. Finally, the method performs the search for ANN performance and parameters simultaneously, thereby constituting a multi-objective search methodology [15,19]. This paper is organized as follows: Section 2 presents some related works; Section 3 presents the bio-inspired methods/algorithms used in the current work; Section 4 describes the proposed method; Section 5 presents the experimental results; and Section 6 summarizes our conclusions and presents future works.

2. Related works

A large number of papers are found in the literature dedicated to designing ANNs automatically through the use of EAs. Cai et al. [9] recently proposed a method that uses a combination of PSO and ES to train Recurrent Neural Networks (RNN) [29] for dealing with times series predictions. With this method, ES performs the global search and PSO performs the local search, refining the solutions. Through an elitism survival selection, half of the existing solutions are then discarded (the losers) and the remaining half (the winners) are used to randomly produce novel solutions. The main problem with this method is that the RNN is trained with a fixed architecture that may not be appropriate for the problem. Moreover, PSO belongs to the EA family, which characteristically performs both global and local searches, but PSO is not built for the local search alone. Therefore, PSO will have a worse performance in most cases than a numerical method built exclusively for such a task [19].

Yu et al. [50] recently proposed a method employing PSO and ES to design ANNs. This method performs the search for the structure (layers, nodes and connections) and parameters (weights and bias). In this case, the PSO performs the global search and the ES performs the local search or fine-tuning of solutions. This method has been experimented with only two problems, demonstrating a satisfactory improvement over the methods used individually with regard to the structure, but not the performance error. In both papers, an EA performs the task normally designed for a numerical method, but, unfortunately, the EA cannot contribute toward finding a better solution than when using a numeral method due to its global exploration aspect applied to the place where a local search mechanism is expected.

Liu et al. [37] propose a memetic algorithm for designing ANNs using a combination of PSO and classic training algorithms, such as the Scaled Conjugated Gradient (SCG) [44]. This method performs the global search using PSO. The training algorithms are then used to fine-tune the weights. In this case, the ANN architecture is fixed and the method searches for the best set of weights and bias that contribute toward a better performance. The method presents satisfactory results, but has been experimented on only three problems instances.

Yao [49] proposes another method employing GAs for optimizing ANNs, in which Evolutionary ANNs (EANNs) are defined as a framework that enables the search for all ANN components needed for its functioning. However, EANNs include a sequential layer search process, in which each layer has specific ANN information to be found by a specific GA, which requires a high computational cost and processing time [4]. Other methods using GAs to optimize a neural network structure are presented in [25,24]. Methods that perform searches including more information, such as transfer functions, initial weights and learning rules (or learning algorithms) are presented in [1,21]. There are also methods that employ non-evolutionary techniques, which prune connections considered less significant [38,31] or freeze weights when the same inputs are submitted to the network [31].

Chen et al. [14] presented a new kind of ANN representation based on trees arrangement which become possible their structural evolution by means of Genetic Programming (GP) and probabilistic incremental program evolution algorithm (PIPE). This framework allows input attributes selection, over-layer connections and different activation functions for different nodes. In some versions the framework proposed by Chen et al. was used mainly to deal with times series predictions [14,13,11], but recently a version to classification problems was shown [12]. In both versions such framework uses EA to train and optimize the ANN structure and performances, but there is no interaction with the traditional numeric learning algorithms. Moreover, the experiments with classification problems need to be extended to include more problems from traditional benchmarks in order to make possible the comparison of the framework against other methods from literature.

In most methods that use GAs to design ANNs, there is no combination with numerical methods for training neural networks. Such a combination may be useful, as the task of optimizing ANNs in order to obtained near-optimal or optimal solutions is very difficult, depending on the problem chosen for the neural network solve [49,1,4]. In other words, the employment of methods specifically built to perform searches considering the entire search space (e.g. global search using GA, ES, PSO, etc), followed by methods for performing the search in a small portion of the search space or fine tuning the solutions (e.g. local search using BP, LM, SCG, etc.) may produce solutions closer to optimality than the use of such methods individually [19,15].

3. Bio-inspired optimization methods

Nature is made up of an immense amount of complex organisms that perform a large number of tasks in order to survive, reproduce and protect their communities. Biology produces complex and self-adaptable organisms that have a vast amount of less reliable components, but possess abilities such as self-assessment, self-repair, self-configuration, levels of redundancy and protection, etc. Examples of such organisms are ants, birds and bees, which individually are fragile, but, together, can be very strong and exhibit intelligence in solving daily problems of survival. As many organisms exhibit very effective functioning, their methodologies and approaches have been extended to solving optimization problems using bio-inspired techniques such as EA and PSO.

GA is one of most popular EAs and is largely applied in optimization problems [19]. The main characteristic of a GA is the presence of mechanisms for selecting and recombining individuals, thereby enabling genetic inheritance from the parents throughout the search execution [19]. Unfortunately, in some problems, such inheritance may not be appropriate or beneficial due to a very complex search space, leading to a more biased exploration/navigation trapped in local maxima/minima [15]. Generating individuals with less inheritance from their parents is a way of preventing the problem of local minima in minimization problems. Moreover, the specification of many parameters is necessary for GA use, such the recombination rate, mutation rate, pressure rate, etc. Due to self-adaptation and having fewer components, ES emerges as a suitable option for use in place of a GA.

Self-adaptation means that some EA parameters are varied during a run in a specific manner: the parameters are included in the individual encoding and co-evolve with the solutions [19]. In contrast to GA, ES only has the mutation, evaluation and initialization mechanisms included in its process, which basically starts with a set of individuals, with the problem information and parameters of the ES encoded together and composing an individual. The population is evaluated. Some individuals are randomly chosen and submitted to the mutation process, in which new individuals are created. If such individuals prove better than their parents, they then replace the parents. Otherwise, new individuals are randomly chosen and the process repeats until reaching a stopping criterion [19]. Thus, there is no survivor selection or recombination of individuals in ES, making the selection less dependent on the parents, which are used to generate new individuals though the mutation process.

An ES is typically used for continuous parameter optimization, with a strong emphasis on mutation for creating offspring and with the mutation parameters changed during a run of the algorithm [19]. The self-adaptation of the ES can be very useful, as the set of parameters required by an optimization method to solve a problem may be unknown or dynamic over time. Thus, GA is not effective for some kinds of problems due to its pre-defined, fixed parameter values [19,15]. On the other hand, the absence of a sophisticated parent selection mechanism makes the search less dependent on the past and more random, which may contribute toward an increase in the time required to find an optimal or near-optimal solution.

PSO is another bio-inspired optimization method. This method was proposed by Kennedy and Eberhart [33] and inspired by the choreography of a flock of birds. The idea of this approach is to simulate the movements of a group (or population) of birds searching for food. The approach can be seen as a distributed behavioral algorithm that performs (in its more general version) a multidimensional search [33,15]. The behavior of each individual (or particle) is affected by either the best local individual (i.e., within a certain neighborhood) or the best global individual. PSO then uses the population concept and a performance measure similar to the fitness value used in EA. Moreover, the adjustments of individuals are analogous to the use of a recombination operator. This approach also introduces the use of flying potential solutions through hyperspace (used to accelerate convergence) [33,15].

PSO allows individuals to benefit from their past experiences, whereas, in an evolutionary algorithm, the current population is normally the only "memory" used by the individuals (e.g. in ES). The advantages of PSO are its simplicity, ease of use and high convergence rate. The disadvantages are mainly related to the apparent difficulties in controlling diversity when used for multi-objective optimization [28,15].

All the optimization methods described above offer both advantages and disadvantages. However, these methods can be combined, employing their individual advantages in order to build a hybrid system that can outperform each individual method. This hybrid system will be applied to a multi-objective optimization problem related to ANNs. Along with the optimization methods described, training algorithms for ANNs are joined to the system, thereby imbuing it with memetic behavior, using optimization methods to perform global searches and numerical methods to perform local searches in the former. Further details on this method are given in the Section 4.

4. The proposed method

The combination of ES, GA, PSO and ANN was performed in order to build a hybrid method capable of seeking near-optimal or even optimal neural networks for a given problem, thereby avoiding the considerable human effort and problems stemming from a manual trial-and-error search. With the use of EAs, an encoding schema and fitness function were defined. The following subsections present these components, the functioning of the EAs and their combination with numerical methods.

4.1. Encoding schema

The proposed method is denominated EAPSONN and has a special encoding schema of solutions that comprises all ANN parameters needed for its functioning. In this encoding schema (Fig. 1), an individual–a representation of a problem solution–contains the ANN information organized in five parts. In the first part, there is information on the learning algorithm type. Such information is stored using the continuous values in which the most representative value indicates what learning algorithm will be used to train the neural network, such as BP, SCG, LM or the quasi-Newton Algorithm (QNA) [22,29]. The use of BP is indicated when the highest values are in the first component of Part 1; SCG is used when the highest values are in the second component and so on. The first part of the structure corresponds to the type of training algorithm used and its size is determined by the number *a* of algorithms included in the search process.

The second part of the data structure involves the parameter values from the learning algorithm specified in the first part. These parameter values are continuous and the length of this part is defined based on the type of learning algorithm. The third part contains information on the hidden layers of the neural network. The length of the third part is fixed. It has four components, the first of which specifies the amount of hidden layers and the others describe the number of hidden nodes per layer. Generally ANN may have up to two hidden layers according recommendations of some authors [7], but networks found by EAPSONN may have up to three hidden layers in order to reduce the number of hidden nodes per layer and investigate the power of an additional layer [4,3]. The fourth part contains information that specifies the type of transfer functions used by each hidden layer, following the same principle as that of the first part. When three types of transfer functions are used in the search process, the fourth part

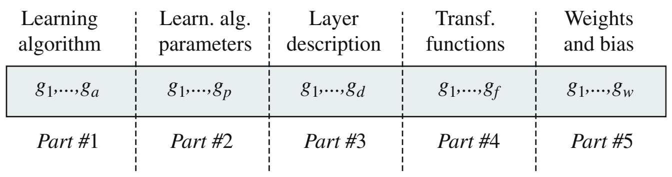

Fig. 1. Encoding schema comprising five parts.

of the data structure is divided into three subparts, each with three components. Each subpart refers to a hidden layer and the highest value within it specifies the type of transfer function of the layer. The fifth part corresponds to the weights and bias in which an array format representing the matrices is stored.

Put simply, an individual can be seen as an array with continuous values representing ANN parameters. This representation makes its manipulation easier with respect to mathematical operations and programming.

4.2. *Fitness function*

For individuals, fitness I_{fit} is composed of five pieces of information: I_{valid}−validation error; I_{train}−training error; I_{hid}−number of hidden layers; I_{nod}−number of hidden nodes; and I_{func}−weight of transfer functions used.

$$I_{fit} = \alpha * I_{valid} + \beta * I_{train} + \gamma * I_{hid} + \delta * I_{nod} + \varepsilon * I_{func} \tag{1}$$

$$I_{nmse} = \frac{100}{NP} * \sum_{j=1}^{P} \sum_{i=1}^{N} (d_i \text{-} o_i)^2 \tag{2}$$

In Eq. (1), I_{valid} and I_{train} are the validating and training Normalized Mean Squared Error (NMSE) generated by the network, respectively; I_{hid} is number of hidden layers (up to 3); I_{nod} represents the total number of hidden nodes; and I_{func} computes the weight of the transfer functions used. For such, each transfer function has an empirically determined associated weight (Table 1): P with 0.2, T with 0.3 and L with 0.5, prioritizing simple transfer functions, as the aim of the EAPSONN is to find simple networks with high performance; N and P are the total number of outputs and number of training patterns, respectively; d and o are the desired output (target) and the network output (obtained), respectively. In this paper, the winner-takes-all classification criterion was adopted, in which the output node with the highest value determines the class pattern. Thus, the number of nodes in the output layer of the neural network is equal to the number of classes of the problem.

Constants $\alpha, \beta, \gamma, \delta$ and ε have values between [0, 1] and control the influence of the respective factors upon the overall fitness calculation process. For example, to favor classification accuracy regarding validating and training, the constants are defined as follows: $\alpha = 0.8, \beta = 0.145, \gamma = 0.03, \delta = 0.005$ and $\varepsilon = 0.02$. These definitions imply that, when apparently similar individuals are found, those that have the least training error, structural complexity and transfer function complexity will prevail. These values were found empirically and used in the EAPSONN experiments.

4.3. *Evolution strategies*

The generation of an initial population is needed in order to begin the functioning of an ES algorithm. In the present paper, a population of n neural networks is created through the random generation of individuals according the schema displayed in Fig. 1. Each individual (a neural network configuration) has an associated self-adaptive parameter vector $v_i, i = 1, \ldots, n$, in which each component serves to control the step size of the search for new mutated parameters of the network. To be consistent with the initialization range, the self-adaptive parameters are initially set to a uniform distribution.

Table 1
List of parameters used by EAPSONN.

Parameters for		Values
EA	Encoding	Direct, using continuous values
	Population size	30
	Elitism	10%
	Pressure of selection	40%
	Mutation	Using (3) and (4)
	Selection criteria	Random tournament with elitism
	Stopping criteria	Max of generations
	Number of generations to ES	50
PSO	Velocity update rule	Using (5)
	Position update rule	Using (6)
	Inertia weight	Uniformly decreasing over time [0.4, 0.9)
	V_{max}	[-0.5, 0.5)
	Number of iterations	10
ANN	Type	Fully connected MLP
	Transfer functions	Pure-linear (P), Tang-sigmoid (T) and Log-Sigmoid (L)
	Hidden layers	Up to 3
	Hidden nodes	Up to 6 per layer
	Training epochs	Up to 5
	Range of initial weights	[-0.5, 0.5)
	Initialization method	Random
	Output neuron	Linear
	Learning and momentum rate	[0.05, 0.25)
	Stopping condition	GL2
	Pruning and Weight decay	Not used
	Sampling method	According to [10]

Each parent generates offspring by varying all the associated information on the neural networks. More specifically, for each parent $P_i, i = 1, \ldots, n$, an offspring P'_i, is created by

$$v'_i(j) = v_i(j).e^{\tau'.N(0,1)+\tau.N(0,1)}, \quad j = 1, \ldots, N_w \tag{3}$$

$$P'_i(j) = P_i(j) + v'_i(j).N(0,1), \quad j = 1, \ldots, N_w \tag{4}$$

in which $\tau' \propto 1/\sqrt{2n}$, $\tau \propto 1/\sqrt{2\sqrt{n}}$, and $N(0,1)$ are random variables obtained from a Normal Distribution, resampled for every j component [19]. The pseudo code of the ES is in Algorithm 1.

Algorithm 1. Pseudo code of Evolution Strategies

```
1   Generate an initial population of individuals with
    random values for ANNs and self-adaptive
    parameters;
2   begin
3     while a sufficiently good fitness
      or iteration is not reached do
4       Evaluate the fitness according the
        given fitness function;
5       Compare the fitness values to find
        the winners through elitism;
6       Keep the winners and produce
        offspring to replace losers
        for the next generation;
        for each offspring do
7         Calculate the self − adaptive
          parameters according to (3);
          Calculate the position according to (4);
8   end
```

As can be seen, the canonical version of the ES algorithm has a simple survivor selection mechanism and the use of this mechanism is not found in some applications. The use of an elitist operator to select survivors and parents for generating offspring may trap the more susceptible individuals in a local minima, which will then be unable to leave such a place in the search space. Due to the fact that this is an important mechanism for the evolutionary search process, a novel selection operator is proposed, with improvements regarding the simplified version of the canonical ES.

In the new selection operator, n individuals are selected using the tournament strategy, with a random pressure rate of $P_{prs}=50\%$ and an elitism rate of $P_{elit}=10\%$ applied to a population of size ω. The tournament strategy [43] was used to create the mating pool and consequently provide the survivors for the next generation. The pseudo code is presented in Algorithm 2. With this algorithm, the maintenance of diversity is ensured due to its random and elitist behavior, selecting both good and not-so-good individuals.

Algorithm 2. Pseudo code of the genetic selection operator

```
1  Create a ranking of individuals considering the fitness;
2  Select the best ($\omega \times P_{elit}$) individuals and place them in the
   matingpool;
3  Update $\omega$ with the number of remaining individuals;
4  Set $currentmember = 1$;
5  begin
6    while $currentmember \leq \omega$ do
7      Pick $k$ individuals randomly, with replacement;
8      Pick a random value $r$ uniformly from $[0,1]$;
9      if $r \leq Pprs$ then
10       Select the worst of these $k$ comparing their fitness;
11     else
12       Select the best of these $k$ comparing their fitness;
13     Denote the selected individual as $i$;
14     Set $mating\_pool[current\_member] = i$;
15     Set $current\_member = current\_member + 1$;
16 end
```

Along with the use of the proposed selection operator, a modification of Eqs. (3) and (4) has been performed, more specifically $N(0,1)$ to $[-1,1)$. The purpose of this modification is to ensure a greater variability in the values that make up an individual and thereby contribute toward greater diversity within a population, together with the proposed selection genetic operator.

4.4. Particle swarm optimization

As mentioned in Section 1, PSO is a kind of bio-inspired optimization and population-based method as well as an ES. Thus, PSO is initialized with a population of random solutions and the search seeks an optimal solution based on a given performance function over iterations/generations. Along with the self-adaptive parameter vector from ES, a randomized velocity vector from PSO is incorporated to the individual data structure shown in Fig. 1. This is due to the fact that each potential solution (*particle*) makes a "flight" through the n-dimensional problem space and, consequently, the particles are assigned a velocity vector, which controls the "flight velocity".

Therefore, each particle i has a position represented by a position vector X_i (using the encoding schema displayed in Fig. 1). A particle swarm moves through an n-dimensional problem space, with the velocity of each particle represented by a vector V_i (random uniform values from [0, 1]). At each iteration, a quality measure is calculated using a function f_i. In the present paper, the fitness function previously described in (1) was used. Moreover, each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector S_b and the best position among all the particles obtained so far in the population is S_g.

For each iteration in time t, using the individual or local best position $S_b(t)$ and global best position $S_g(t)$, a new velocity vector for particle i is calculated using (5), in which c_1 and c_2 are positive acceleration constants; ϕ_1 and ϕ_2 are uniformly distributed random numbers in the range [0,1]; and w is the inertia weight, starting with 0.9 and decreasing uniformly to 0.4.

$$V_i(t+1) = w \times V_i(t) + c_1\phi_1(S_b(t)\text{-}X_i(t)) + c_1\phi_2(S_g(t)-X_i(t)) \quad (5)$$

The term V_i is limited to the range $\pm V_{max}$. Changing the velocity in this way enables particle i to search for its individual best position S_b and the global best position S_g. Based on the updated velocities, each particle changes its position according to (6).

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (6)$$

Using (5) and (6), the population of particles tends to cluster together, moving in a random direction until a stopping criterion is reached. The generic pseudo code of the PSO is given in Algorithm 3.

Algorithm 3. Pseudo code of Particle Swarm Optimization

```
1  Initialize a population of particles
   with random position and velocities in the
   $n$-dimensional problem space;
2  begin
3    while a sufficiently good fitness or a
         maximum number of iterations is not reached do
4      Evaluate the fitness according to
       some fitness function;
5      Update the $S_b$ if fitness value of the current
       particle is better than $S_b(t\text{-}1)$;
6      Determine $S_g(t)$, choose the particle with
       the best fitness values;
7      for each particle do
8        Calculate the new velocity of the particle
           according to (5);
         Calculate new position of the
         particle according to (6);
9  end
```

In EAPSONN, an implementation of PSO was used that relies on the usage of the fitness function described in (1), which is calculated with the same individual representation displayed in Fig. 1 (also used by the ES algorithm). The V_{max} value starts with 0.9 and is uniformly decreased down to 0.4. As seen from its pseudo code, there is no selection operator for parents and survivors in PSO. Due to the absence of a mechanism for the replacement of bad particles/individuals (e.g. particles that do not contribute toward finding optimal solutions and trap the algorithm in a local minima), the PSO algorithm suffers with a loss of diversity and can consequently be trapped in a local minima. To avoid these problems, a combination of ES, GA and PSO is proposed in the next subsection.

4.5. Hybridization of ES, GA, PSO and ANN—EAPSONN

ES usage in optimization problems is motivated mainly by the presence of parameters that are self-adaptive over time based on the surface of the search space; are easy to implement; and have a history of success in continuous problem optimization. However, there is no selection of parents and survivors through a sophisticated genetic selection operator. GA provides a large range of genetic operators for the recombination, mutation and selection of individuals, but its parameters are normally fixed at the start of the search process and do not change over time as in ES. However, a combination of ES and GA can be performed to circumvent the absence of the selection operator in the former and absence of the self-adaptive parameters in the latter. With such a combination, the global search aspect is maintained, but a local search for fine tuning the solutions/individuals may be needed to find near-optimal or even optimal solutions.

PSO is a frequently used bio-inspired optimization methods for performing local searches due to the social and cognitive adaptation among the particles, focusing more on cooperation. PSO normally has a better performance than ES and GA when applied to perform local searches in a portion of an entire search space. Unfortunately, PSO has drawbacks, such as the loss of diversity, no selection operation and, in some cases, it is inefficient at performing a satisfactory local search that returns near-optimal solutions. Some of the problems found in bio-inspired optimization methods can be solved by combining methods, as they have complementary characteristics. However, these methods are not useful or the exploitation task (local search) due to the fact that they are directed toward the exploration task, e.g. a global search that considers the entire search space.

To perform the exploitation task with more effectiveness, BP and LM, which are well-known training algorithms for neural networks, were combined. Thus, the hybridization of ES and GA has the advantage of combining the power of self-adaptation with genetic operators for selection of parents and survivors; after that, the addition of PSO brings the capability to perform local searches and its execution speed. Finally, the integration of numerical methods was performed to improve the quality of local searches. In this way the combination of ES, GA and PSO is responsible for executing the exploration of the search space and starting the exploitation task with PSO. This task is then completed with the use of BP or/and LM training algorithms. The EAPSONN pseudo code is summarized in Algorithm 4.

Algorithm 4. Pseudo code of the EAPSONN

```
1  Initialize a population of individuals with random
   values using the data structure given in Fig. 1, random
   velocities and self-adaptive parameters in an
   n-dimensional problem space;
2  Apply learning algorithms BP/LM and evaluate the
   fitness according to (1);
3  begin
4    while the max number of iterations is not reached do
5      Use 0.33emAlgorithm 2 to select parents/survivors (winners);
6      Mutate winners to produce children using (3) with $N(-1,1)$;
7      Update self – adaptive parameter vector using (4) for each child;
8      Enhance the children with PSO up to some number of iterations;
9      for each child/particle do
10       Update $S_b$ if the current fitness value of the particle is better than $S_b$;
11       Determine $S_g$, choose the particle with the best fitness values of all;
12       Calculate the new velocity of the particle according to (5);
13       Calculate new position of the particle according to (6);
14     Apply learning algorithms BP/LM to the enhanced children and complete the enhancement phase;
15     Evaluate the fitness of the children according to (1);
16     Merge children and winners to form the offspring;
17 end
```

Table 1 summarizes all parameters/information related to the EAPSONN method. This information refers to the ranges and rates used to optimize the parameters and performance of neural networks.

The ANN information displayed in Table 1 refers to the possible values that neural networks found by EAPSONN can take. Parameter values for EA and PSO were found empirically through many runs of EAPSONN and produce the best results. The next section presents the experiments, results and discussion on the proposed method.

5. Experiments

Classification problems from different domains were used in experiments comparing the performance of the EAPSONN to other methods found in the literature. Statistical tests and an improved strategy were used in the experiments in order to make the results more reliable. The experiments were performed with eight well-known classification problems found in the UCI repository [5] and listed in Table 2, presenting the number of attributes (AT), classes (CL), training examples (TR), validation examples (VA) and test examples (TE).

Table 2
Summary of data sets used in the experiments.

Name	Composition				
	TR	VA	TE	AT	CA
Cancer	228	122	349	9	2
Glass	72	37	105	9	6
Heart-Cleveland	98	54	151	35	2
Heart	300	161	459	35	2
Horse	119	63	182	58	3
Pima-Diabetes	249	135	384	8	2
Card	225	121	344	51	2
Soybean	219	124	340	82	19

There exists a large kind of strategies for design experimentations of ANNs as those presented by [30,34]. Specifically to current work was used the strategy studied in recent work of Cant-Paz and Kamath [10] which is devoted to comparisons of combinations of EA and ANN for classification problems. The strategy of Cant-Paz and Kamath [10] was preferred due to it simplicity, largely used and consequently make easier repeat the experiments that will be reported in this section. In this way, was used 30 two-fold iterations. At each iteration, data were randomly divided into halves. One half was the input for the algorithm (65% for training and 35% for the validation set) and the other half was used to test the final solution (test set). The execution of one iteration corresponds to the creation of an initial population and execution of an evolutionary search over 50 generations; in each generation, PSO is executed 10 times and the weights of the children are updated every 5 epochs, using the BP/LM learning algorithms. After 30 iterations with different data divisions and initial populations, the best 30 ANN parameters are chosen based on the fitness function (1). The results reported are the mean value from the 30 ANN parameters found for each classification problem.

A search for ANNs through trial-and-error was performed following the previously described methodology, using the same database split schema and ANN information displayed in Table 1. We performed 30 runs in each fold for the networks created randomly with the information described in Table 1 and selected using the proposed genetic operator for the selection of survivors. In this simulation, the randomly generated neural networks are evaluated according to the fitness function (1) using BP and a combination of BP-LM. The winners persist to the next iteration and a new set of neural networks is generated and evaluated again until the maximal number of iterations.

ES and PSO were also applied separately in the optimization of ANNs to the previously generated data folds. Both methods were applied using the proposed genetic operator for the selection and fitness function (1). However, differently of the trial-and-error method, the learning algorithm was not used for updating the weights in the execution of ES and PSO. To determine whether the differences among the methods were statistically significant, we used the *T*-test hypothesis test with significance level 5%.

The data sets sampling is replicated for each version of the methods considered in experiments and for each method their populations are randomized for each data fold. Tables 3–6 display the results obtained from the trial-and-error simulation (TAE) using BP and LM algorithms and EAPSONN using BP (EN-BP) and using BP and LM together (EN-BPLM), as well as results using ES and PSO separately. For each method (M) the results are shown the average, standard deviation, maximum and minimum values. The **boldface** and *emphasized* values meaning that they are better than all or better than trial-and-error, PSO and ES according to the statistic *T*-test, respectively.

Information related to composition/structure of ANNs are listed according to the average number of hidden layers (HL) and the correspondent average number of hidden nodes (HN), learning rate parameters and errors. EAPSONN outperformed ES, PSO and the trial-and-error method regarding to the mean error performance of the ANNs found using BP for the Cancer problem (Table 3). With the use of the LM algorithm, the proposed method was able to obtain the best results only in the learning and validation phases, whereas its performance in the test and classification phases was not statistically different from the simulation with the trial-and-error method. For the Glass problem, the ANNs found with EAPSONN (using BP and/or LM) were the best with regard to learning, validation and classification errors, but required a greater number of nodes than the other methods.

Table 4 displays the results of the Heart-Cleveland and Heart problems. For the first problem, the ANNs found with EAPSONN using the BP algorithm were better than those found by the other search methods using the same learning algorithm. For the Heart-Cleveland problem, the results found with the trial-and-error method using the LM algorithm were the best in terms of test and classification errors. The EAPSONN method using both the BP and LM algorithms achieved the best performance in terms of learning and validation errors, but this did not lead to an improvement regarding to the test and classification errors. An analysis of the results of the Heart problem reveals that the performances of the search methods were similar to those for the Heart-Cleveland problem. For both problems, EAPSONN found ANNs with an average number of nodes similar to the results using the other search methods.

Table 5 displays the ANNs found for the Pima-Diabetes problem. Considering the search methods without the use of the LM algorithm, EAPSONN obtained the best results in terms of errors. Using the LM algorithm, the proposed method achieved the best results regarding to the learning and validation data, whereas the simulation of the trial-and-error method achieved the best results regarding to the test and classification data. For the Horse problem, the best ANNs in terms of errors were found with the simulation of the trial-and-error method using the LM algorithm. However, when using the BP algorithm, EAPSONN achieved better results than the ES, PSO and trial-and-error method using this same learning algorithm.

The values displayed in Table 6 reveal that the proposed method using the BP algorithm achieved the best results among the search methods that did not employ the LM algorithm. An analysis of the versions that employed the LM algorithm reveals that EAPSONN achieved the best performance in the learning and validation sets, whereas the trial-and-error methods achieved the best performance regarding the test set and classification. For the Soybean problem, EAPSONN achieved the best results using the BP algorithm as well as using a combination of the BP and LM algorithms.

In an overall analysis, the EAPSONN method—which is a combination of evolutionary search methods together with numerical methods—proved to be very efficient in finding near-optimal ANNs. The version of the proposed method that employs the BP algorithm achieved better results for all the problems in comparison to the other search methods used separately (trial-and-error, ES and PSO). In some cases, EAPSONN failed to outperform the trial-and-error method using LM. This occurred because LM is a second-order algorithm that converges faster than first-order methods and does not experience as much interference from the variation in the number of hidden layers and neurons, type of transfer function or initialization weights as first-order algorithms do [35,20,48,4]. As EAPSONN works with the variations in parameters in order to achieve better results

Table 3
Results for Cancer and Glass problems.

M	Learn. rate	HL	HN	Train. error	Valid. error	Test error	Classif. error
Cancer							
TAE-BP	0.1523	1.3	5.6	0.1185	0.1185	0.1197	0.0861
	0.4566	0.5	2.6	0.0285	0.0264	0.0304	0.0459
	0.2432	2	11	0.1651	0.1587	0.1700	0.2034
	0.0544	1	1	0.0532	0.0571	0.0572	0.0287
TAE-LM	0.0947	3	5.9	0.0211	0.0263	**0.0282**	**0.0346**
	0.0744	0	1.6	0.0064	0.0093	0.0045	0.0074
	0.25	3	8	0.0359	0.0460	0.0371	0.0458
	0.05	3	3	0.0085	0.0042	0.0191	0.0143
ES	0.1822	1.4	5.8	0.1013	0.1007	0.1014	0.0587
	0.0565	0.6	2.4	0.0294	0.0284	0.0290	0.0305
	0.25	3	13	0.1508	0.1438	0.1501	0.1691
	0.0775	1	2	0.0326	0.0380	0.0386	0.0258
PSO	0.2041	1.9	6.9	0.0673	0.0620	0.0638	0.0542
	0.0749	0.6	2.6	0.0194	0.0168	0.0156	0.0190
	0.5015	3	13	0.0954	0.0912	0.0941	0.0917
	0.0773	1	1	0.0277	0.0277	0.0341	0.0229
EN-BP	0.1817	2.8	8.9	*0.0334*	*0.0274*	*0.0344*	*0.0387*
	0.0602	0.4	2.2	0.0110	0.0107	0.0085	0.0110
	0.2471	3	13	0.0611	0.0521	0.0531	0.0630
	0.0515	2	5	0.0172	0.0073	0.0213	0.0143
EN-BPLM	0.1690	3	6.6	**0.0133**	**0.0163**	0.0396	0.0449
	0.0525	0	1.5	0.0090	0.0100	0.0122	0.0138
	0.2490	3	10	0.0331	0.0378	0.0720	0.0860
	0.0576	3	4	0	0	0.0184	0.0229
Glass							
TAE-BP	0.0860	2.8	6.6	0.1248	0.1241	0.1221	0.6406
	0.0486	0.5	2.4	0.0030	0.0034	0.0022	0.0170
	0.2500	3	11	0.1312	0.1304	0.1264	0.6667
	0.0500	1	2	0.1132	0.1095	0.1120	0.5714
TAE-LM	0.0560	2.9	8.1	0.0897	0.0944	**0.0995**	0.4381
	0.0275	0.2	2.8	0.0070	0.0053	0.0041	0.0533
	0.2004	3	15	0.1029	0.1046	0.1110	0.5810
	0.0500	2	3	0.0760	0.0813	0.0944	0.3810
ES	0.2067	3	9.8	0.1253	0.1247	0.1228	0.6425
	0.0527	0	2.2	0.0014	0.0013	0.0013	0.0119
	0.2500	3	14	0.1294	0.1287	0.1272	0.6857
	0.0569	3	5	0.1241	0.1235	0.1217	0.6381
PSO	0.0876	2.7	7.9	0.1246	0.1235	0.1223	0.6429
	0.0492	0.5	2.6	0.0031	0.0028	0.0028	0.0310
	0.2500	3	14	0.1305	0.1272	0.1265	0.7429
	0.0500	1	1	0.1154	0.1162	0.1154	0.5619
EN-BP	0.1920	2.9	7.6	*0.1192*	*0.1171*	*0.1192*	*0.6210*
	0.0459	0.2	2.1	0.0052	0.0049	0.0048	0.0510
	0.2495	3	11	0.1257	0.1241	0.1352	0.7524
	0.0938	2	3	0.1057	0.1075	0.1102	0.5333
EN-BPLM	0.1668	2.9	10.6	**0.0514**	**0.0734**	0.1070	**0.4184**
	0.0558	0.3	2.4	0.0188	0.0095	0.0096	0.0421
	0.2435	3	15	0.0809	0.0914	0.1279	0.4952
	0.0631	2	5	0.0125	0.0568	0.0871	0.3429

(networks), good results are not always achieved when using a combination of second-order and first-order algorithms.

The simulation of the trial-and-error method using the LM algorithm, managed to obtain the best performance in terms of test error and classification for most problems used in experiments. However, the use of the LM algorithm in training RNAs offers some disadvantages, such as: the need for a large memory for matrix operations in each iteration and the computational complexity increases with the number of weights in a quadratic manner. One consequence of these disadvantages is that the search methods which use the algorithm LM are slower, due to the need to execute more complex calculations and also to allocate and misallocate a larger part of memory than methods which use algorithms of first order training, like the BP algorithm. In this way, even with errors not considerably better than the methods that use LM, the EAPSONN using the BP algorithm, is a good intermediate option as it brings together positive features such as: less search time compared with the LM algorithm version; for most problems it can find networks with a lower average number of hidden layers and nodes; is superior to other methods which use the BP algorithm, getting close to methods which use the LM algorithm, considering the size of the errors.

It is important to mention that the good performance of the trial-and-error simulation, using BP and LM algorithms is also due to the use of the genetic selection operator proposed in this work. Normally in the work of manual search for neural networks configurations does not observe the use of advanced mechanisms of selection of survivors for the next iterations, as happens in this work. In this way, the simulation of trail and error is a method of automatic search, although simplified and which does not faithfully reflect the manual search process. Thus, the manual search process tends to be slower, more tiring and perhaps less productive.

Table 4
Results for Heart-Cleveland and Heart problems.

M	Learn. rate	HL	HN	Train. error	Valid. error	Test error	Classif. error
Heart-Cleveland							
TAE-BP	0.2156	1.27	5.6	0.1813	0.1726	0.1856	0.2404
	0.0300	0.45	2.51	0.0185	0.0190	0.0197	0.0377
	0.2500	2	11	0.2218	0.2128	0.2185	0.3113
	0.1211	1	2	0.1356	0.1361	0.1417	0.1722
TAE-LM	0.0826	2.77	8.83	0.0731	0.0996	**0.1441**	**0.1852**
	0.0726	0.43	3.05	0.0234	0.0264	0.0210	0.0250
	0.2500	3	16	0.1230	0.1506	0.1910	0.2450
	0.0500	2	3	0.0320	0.0503	0.1021	0.1457
ES	0.1810	1.8	6.47	0.2121	0.2115	0.2138	0.3071
	0.0494	0.66	2.83	0.0254	0.0239	0.0246	0.0921
	0.2470	3	14	0.2497	0.2490	0.2484	0.4570
	0.0549	1	2	0.1600	0.1609	0.1547	0.1523
PSO	0.1910	2.03	7.7	0.1758	0.1563	0.1784	0.2298
	0.0564	0.77	4.16	0.0228	0.0260	0.0214	0.0394
	0.2500	3	15	0.2179	0.2007	0.2198	0.3046
	0.0500	1	1	0.1353	0.0890	0.1396	0.1258
EN-BP	0.1855	2.17	7.6	*0.1402*	*0.1212*	*0.1597*	*0.2188*
	0.0585	0.79	3.05	0.0178	0.0232	0.0225	0.0365
	0.2484	3	13	0.1771	0.1705	0.2090	0.3046
	0.0631	1	3	0.0956	0.0726	0.1277	0.1589
EN-BPLM	0.1429	2.73	7.9	**0.0447**	**0.0687**	0.1862	0.2064
	0.0522	0.45	2.25	0.0293	0.0262	0.0387	0.0288
	0.2380	3	13	0.1095	0.1205	0.3051	0.2715
	0.0550	2	4	0	0.0193	0.1339	0.1589
Heart							
TAE-BP	0.1867	1.57	6.5	0.1026	0.1423	0.1476	0.2033
	0.0507	0.5	2.57	0.0169	0.0203	0.0155	0.0281
	0.2500	2	10	0.1504	0.1803	0.1970	0.2963
	0.0500	1	2	0.0697	0.1032	0.1269	0.1590
TAE-LM	0.1299	2.97	7.2	0.0943	0.1237	**0.1411**	**0.1946**
	0.0936	0.18	2.02	0.0126	0.0162	0.0075	0.0154
	0.2500	3	11	0.1283	0.1603	0.1567	0.2288
	0.0500	2	3	0.0594	0.0758	0.1277	0.1634
ES	0.2419	2.37	5.57	0.2305	0.2303	0.2311	0.3733
	0.3333	0.81	2.28	0.0196	0.0204	0.0192	0.0840
	1.9882	3	12	0.2495	0.2495	0.2495	0.4466
	0.0559	1	1	0.1858	0.1865	0.1848	0.2222
PSO	0.2101	1.97	7.5	0.1832	0.1845	0.1944	0.2649
	0.0427	0.62	2.7	0.0158	0.0157	0.0149	0.0322
	0.2500	3	12	0.2164	0.2172	0.2320	0.3290
	0.0934	1	2	0.1526	0.1559	0.1677	0.2157
EN-BP	0.1808	2.13	6.97	*0.0905*	*0.1322*	*0.1465*	*0.2005*
	0.0554	0.82	3.02	0.0172	0.0151	0.0117	0.0165
	0.2496	3	14	0.1295	0.1807	0.1660	0.2331
	0.0534	1	1	0.0637	0.1021	0.1244	0.1656
EN-BPLM	0.1738	2.87	7.63	**0.0716**	**0.1155**	0.1597	0.2065
	0.0472	0.35	2.55	0.0226	0.0184	0.0159	0.0195
	0.2456	3	13	0.1141	0.1481	0.1837	0.2484
	0.0738	2	3	0.0272	0.0572	0.1333	0.1678

An analysis of the number of hidden nodes reveals that the ANNs found with the EAPSONN method had a greater number of nodes than the ANNs found with the other search methods. However, the results of the EAPSONN method using the BP algorithm were the best for all problems when comparing the other methods separately using this algorithm in the search for near-optimal ANNs. Furthermore, for the Glass and Soybean problems, the best results were achieved with the proposed method employing both the BP and LM algorithms. This demonstrates the efficiency of combining different optimization methods responsible for maintaining populations of individuals with considerable diversity, thereby allowing different points of the search space to be explored and contributing toward a better overall performance than methods that do not employ this combination. The capability of the proposed method in finding high-performance ANNs and an acceptable number of hidden layers and nodes is evident. In the special case of using the BP algorithm, EAPSONN achieved much better results in terms of errors when compared to the trial-and-error, ES and PSO methods used separately.

Table 7 compares the results obtained with EAPSONN and other methods found in the literature. Comparisons between

Table 5
Results for Pima and Horse problems.

M	Learn. rate	HL	HN	Train. error	Valid. error	Test error	Classif. error
Pima-Diabetes							
TAE-BP	0.2476	2.8	6.73	0.2263	0.2257	0.2264	0.3468
	0.2008	0.61	1.87	0.0064	0.0069	0.0055	0.0078
	1.2660	3	10	0.2329	0.2326	0.2328	0.3490
	0.0500	1	3	0.2038	0.2010	0.2040	0.3099
TAE-LM	0.0905	2.93	7.23	0.1438	0.1529	**0.1639**	**0.2375**
	0.0626	0.25	1.83	0.0100	0.0163	0.0087	0.0174
	0.2192	3	11	0.1609	0.1926	0.1870	0.2943
	0.0500	2	3	0.1259	0.1129	0.1511	0.2135
ES	0.1901	2.73	5.27	0.2219	0.2213	0.2228	0.3425
	0.0524	0.64	1.55	0.0121	0.0128	0.0108	0.0200
	0.2489	3	11	0.2285	0.2281	0.2284	0.3490
	0.0533	1	3	0.1837	0.1838	0.1868	0.2526
PSO	0.1568	2.23	7.83	0.2039	0.1981	0.2039	0.3066
	0.0611	0.73	3.31	0.0134	0.0127	0.0135	0.0281
	0.2471	3	15	0.2443	0.2214	0.2332	0.3490
	0.0180	1	2	0.1808	0.1717	0.1788	0.2500
EN-BP	0.1632	2.97	8	*0.1277*	*0.1454*	*0.1836*	*0.2533*
	0.0492	0.18	1.91	0.0186	0.0219	0.0156	0.0189
	0.2473	3	12	0.1898	0.2264	0.2230	0.2943
	0.0729	2	3	0.1003	0.1065	0.1639	0.2135
EN-BPLM	0.1445	2.97	7.97	**0.1185**	**0.1413**	0.1832	0.2561
	0.0576	0.18	2.08	0.0191	0.0151	0.0180	0.0202
	0.2494	3	11	0.1496	0.1777	0.2351	0.2969
	0.0506	2	4	0.0756	0.1053	0.1571	0.2083
Horse							
TAE-BP	0.0991	2.93	8	0.1814	0.1800	0.1815	0.3863
	0.0643	0.25	1.74	0.0057	0.0045	0.0036	0.0081
	0.2500	3	11	0.1874	0.1857	0.1865	0.4286
	0.0500	2	4	0.1569	0.1642	0.1663	0.3846
TAE-LM	0.1355	2.83	9.3	**0.0752**	0.1411	**0.1669**	**0.3663**
	0.0856	0.38	2.2	0.0287	0.0125	0.0124	0.0368
	0.2500	3	14	0.1280	0.1586	0.2013	0.4451
	0.0500	2	5	0.0261	0.1055	0.1490	0.2967
ES	0.2174	3	6.3	0.1816	0.1802	0.1811	0.3846
	0.0320	0	1.12	0.0004	0.0004	0.0004	0
	0.2500	3	8	0.1832	0.1817	0.1825	0.3846
	0.1036	3	4	0.1812	0.1797	0.1802	0.3846
PSO	0.5248	2.83	8.23	0.1816	0.1771	0.1814	0.3866
	0.7547	0.46	2.86	0.0048	0.0068	0.0045	0.0053
	2.8960	3	15	0.1880	0.1845	0.1906	0.4066
	0	1	2	0.1662	0.1552	0.1689	0.3846
EN-BP	0.1921	2.87	5.57	*0.1440*	*0.1668*	*0.1790*	*0.3844*
	0.0555	0.43	2.25	0.0409	0.0145	0.0091	0.0220
	0.2459	3	10	0.1813	0.1800	0.2013	0.4725
	0.0645	1	3	0.0726	0.1341	0.1584	0.3407
EN-BPLM	0.1545	2.97	7.93	0.0869	**0.1257**	0.2728	0.3938
	0.0573	0.18	2.08	0.0328	0.0161	0.4491	0.0332
	0.2323	3	13	0.1354	0.1546	2.6483	0.4560
	0.0642	2	4	0.0055	0.0926	0.1565	0.3187

these methods must be made with caution, as the results are obtained with different experimental model setups, but errors are estimated with the same method. Nonetheless, EAPSONN proved to be better at finding ANNs with few hidden nodes and is situated among the methods that find neural networks with the lowest mean error.

Most methods found in the literature related to ANN optimization fail to describe time consumption. Comparisons of this variable are often difficult due to the use of different computer configurations, operational systems, etc. Table 8 displays information on the time EAPSONN required to find ANNs for each problem using all leaning algorithms. Time is related to the search performed in one fold of the each problem, measured in minutes of processing on a computer with Microsoft Windows operational system, 3.5 GB of RAM and a processor Intel Pentium of 3 GHz, using the Matlab tool.

From the figures in Table 8, the EAPSONN method required more time to optimize neural networks for problems with a high number of attributes (input values). Using the LM and BP algorithms together, EAPSONN required more time than when using just the BP algorithm. In the special case of Soybean problem, the proposed method required a huge amount of time to

Table 6
Results for Card and Soybean problems.

M	Learn. rate	HL	HN	Train. error	Valid. error	Test error	Classif. error
Card							
TAE-BP	0.2208	1.37	5.43	0.1999	0.1984	0.2024	0.2712
	0.0278	0.49	2.43	0.0214	0.0185	0.0206	0.0690
	0.2500	2	11	0.2369	0.2306	0.2420	0.4390
	0.1427	1	2	0.1612	0.1616	0.1661	0.1773
TAE-LM	0.1226	2.83	7.13	0.0717	0.0991	**0.1132**	**0.1481**
	0.0867	0.38	2.22	0.0140	0.0166	0.0124	0.0157
	0.2500	3	11	0.0967	0.1394	0.1481	0.1831
	0.0500	2	3	0.0398	0.0682	0.0919	0.1076
ES	0.1880	2.57	6.07	0.2336	0.2348	0.2349	0.3981
	0.0482	0.68	2.79	0.0194	0.0166	0.0159	0.0774
	0.2466	3	14	0.2479	0.2481	0.2479	0.4448
	0.0607	1	1	0.1775	0.1842	0.1877	0.1773
PSO	0.3126	2.13	8.07	0.1968	0.1950	0.1995	0.2652
	0.4005	0.63	3.35	0.0203	0.0204	0.0231	0.0691
	2.1057	3	16	0.2346	0.2226	0.2410	0.3924
	0.0148	1	2	0.1535	0.1429	0.1518	0.1686
EN-BP	0.1695	2.27	7.13	*0.1462*	*0.1423*	*0.1582*	*0.2085*
	0.0567	0.74	2.78	0.0237	0.0266	0.0405	0.0456
	0.2458	3	12	0.1980	0.2158	0.3368	0.3634
	0.0682	1	2	0.0880	0.0880	0.1225	0.1453
EN-BPLM	0.1624	2.97	7.73	**0.0554**	**0.0834**	0.1417	0.1624
	0.0446	0.18	1.82	0.0223	0.0175	0.0296	0.0186
	0.2407	3	11	0.1053	0.1147	0.2795	0.2006
	0.0630	2	4	0.0123	0.0449	0.1153	0.1308
Soybean							
TAE-BP	0.1118	3	6.77	0.0486	0.0490	0.0486	0.8550
	0.0715	0	1.77	0.0009	0.0009	0.0008	0.0350
	0.2500	3	11	0.0513	0.0517	0.0515	0.9088
	0.0500	3	4	0.0477	0.0481	0.0476	0.7471
TAE-LM	0.0670	3	8.07	0.0361	0.0379	0.0372	0.6017
	0.0530	0	2.72	0.0038	0.0033	0.0033	0.0916
	0.2500	3	15	0.0422	0.0430	0.0425	0.7471
	0.0500	3	4	0.0267	0.0303	0.0294	0.4000
ES	0.1545	2.77	8.93	0.0516	0.0520	0.0518	0.8703
	0.0687	0.43	2.58	0.0041	0.0041	0.0040	0.0127
	0.2500	3	13	0.0639	0.0642	0.0640	0.9353
	0.0594	2	4	0.0479	0.0483	0.0481	0.8647
PSO	0.0822	2.9	7.13	0.0493	0.0496	0.0493	0.8638
	0.0546	0.31	2.11	0.0018	0.0016	0.0015	0.0506
	0.2500	3	11	0.0542	0.0540	0.0542	1
	0.0500	2	3	0.0474	0.0477	0.0476	0.7176
EN-BP	0.1733	2.97	7.13	*0.0472*	*0.0477*	*0.0475*	*0.8446*
	0.0621	0.18	2.03	0.0014	0.0012	0.0012	0.0463
	0.2500	3	12	0.0479	0.0483	0.0480	0.8676
	0.0599	2	3	0.0427	0.0435	0.0432	0.7059
EN-BPLM	0.0788	2.77	9.6	**0.0287**	**0.0330**	**0.0338**	**0.5274**
	0.0607	0.5	1.9	0.0107	0.0086	0.0085	0.1593
	0.2500	3	12	0.0479	0.0483	0.0508	0.8647
	0.0500	1	5	0.0175	0.0226	0.0228	0.3618

optimize ANNs. This behavior may be due to the high number of attributes and classes in the Soybean problem and the fact that second-order algorithms require complex calculus to adjust the weights and, consequently, require more memory and processing time.

The variations of the trial-and-error method, such as the ES method, require a much lower search time compared to other methods. This is due to the fact that such methods execute few calculus numbers to find near-optimal networks, mainly based on randomly adding the search process without worrying excessively about the strategy adopted for this addition.

6. Conclusions and future works

This paper proposed a novel method for optimizing the parameters and performance of ANNs applied to standard supervised classifications problems. The method, denominated EAPSONN, is made up of a combination of ES, GA, PSO and learning algorithms that are widely used in studies involving ANNs. Due to its particular combination of these methods, EAPSONN is an original, unique method that aggregates the characteristics of memetic, hybrid and multi-objective search methods designed to search for MLP ANNs. In order to assess the

Table 7
Comparison with other methods in the literature.

Information	Problem	Methods								
		EAPSONN	GANNTune [3]	NNGA-DCOD [4]	GEPNET [25]	COVNET [25]	MOBNET [25]	COOPNN [24]	ESPNet [50]	Immune ensembles [23]
Test Error	Cancer	3.44	3.63	6.22	–	–	–	1.38	–	30.14
	Glass	11.92	10.27	12.75	35.16	–	35.16	22.89	–	25.66
	Heart-C	15.97	14.93	14.26	13.63	14.26	13.63	11.96	—	11.84
	Heart	14.65	–	–	–	–	–	–	–	13.82
	Horse	18.32	17.79	17.52	–	–	–	26.74	–	28.24
	Pima	17.9	16.99	21.15	19.27	19.90	19.84	19.69	23.81	23.75
	Card	14.17	12.35	–	–	–	–	12.17	13.48	12.44
	Soybean	3.38	4.79	–	–	–	–	7.61	–	9.88
Nodes	Cancer	8.9	25.4	12.8	–	–	–	5.89	–	11
	Glass	7.6	27.3	5.6	6.33	–	14.87	6.73	–	14
	Heart-C	7.6	20.5	12.1	6.37	4.77	11.4	7.28	–	16
	Heart	7.5	–	–	–	–	–	–	–	12.2
	Horse	8	24.5	7.2	–	–	–	20.3	–	18.2
	Pima	5.6	26.6	6.1	4.57	6.17	7.9	7.9	3.3	6.4
	Card	7.7	18.6	–	–	–	–	6.89	3.2	10
	Soybean	9.6	38.9	–	–	–	–	19.42	–	13.6

Table 8
Mean time in minutes of processing to searching near-optimal ANNs with following methods: trial-and-error (TAE) using BP and LM algorithms, ES, PSO and EPSONN using BP and a combination of BP and LM algorithms.

Problem	TAE	TAE	ES	PSO	EAPSONN	
	BP	LM			BP	BP and LM
Cancer	3.42	4.02	3.48	37.46	35.92	40.35
Glass	3.38	4.11	3.62	36.68	35.67	42.31
Heart-C	3.39	4.73	3.69	36.10	34.67	45.21
Heart	3.48	6.17	3.91	38.50	38.24	55.71
Horse	3.46	7.86	3.86	36.61	37.62	66.98
Pima	3.45	4.01	3.83	37.21	40.88	40.93
Card	3.47	7.42	3.98	38.06	36.59	63.53
Soybean	3.58	86.28	4.03	38.30	38.41	466.16

performance of the proposed method, experiments were carried out with different database types and dimensions and comparisons were made with other methods found in the literature designed for the optimization of ANNs. An analysis of the time EAPSONN required to find solutions was also carried out.

Based on the results of the experiments, EAPSONN using the BP algorithm is capable of finding ANNs with better parameters and performance than the ES, PSO and a simulation of the manual trial-and-error method for all problems. Using a combination of the BP and LM algorithms, the proposed method achieved a better performance in only two problems when compared to the trial-and-error method using the LM algorithm. This may be explained by the fact that purely second-order search methods achieve a better performance than a combination of first-order and second-order methods. When compared to other methods in the literature, EAPSONN is situated among those that have the best performance for all these problems, obtaining ANNs with a lower number of nodes. Such aspects discussed and due to be a novel option to automatic design of ANNs comprising the advantages of EAPSONN. On the other hand, one disadvantage of the proposed method is the considerable time required for the execution of the search. This may be explained by the different combined search methods, which explore the search space in different ways and therefore require more time.

Thus, EAPSONN has the capacity to optimize nearly all ANN parameters needed for its functioning. Unfortunately, there are disadvantages such as the time required to execute the search and the fact that the combination of the BP and LM algorithms did not achieve the expected results. However, with the execution of further studies, considerable improvements in the method are expected, such as the following: a reduction in execution time through a revision of the strategies adopted in each search method; tests with other forms of combinations; the use of other learning algorithms; and the use of other evolutionary search methods.

Acknowledgments

The authors would like to thank CNPq, CAPES and FACEPE (Brazilian Research Agencies) for their financial support.

References

[1] A. Abraham, Meta learning evolutionary artificial neural networks, Neurocomputing 56 (2004) 1–38.
[2] L.M. Almeida, T.B. Ludermir, Automatically searching near-optimal artificial neural networks, in: Proceedings of the European Symposium on Artificial Neural Networks (ESANN'07), 2007, pp. 549–554.
[3] L.M. Almeida, T.B. Ludermir, An evolutionary approach for tuning artificial neural network parameters, in: Proceedings of the Third International Workshop on Hybrid Artificial Intelligence Systems (HAIS'08), 2008, pp. 156–163.
[4] L.M. Almeida, T.B. Ludermir, An improved method for automatically searching near-optimal artificial neural networks, in: IEEE International Joint Conference on Neural Networks (IJCNN'08) (IEEE World Congress on Computational Intelligence), 2008, pp. 2235–2242.
[5] A. Asuncion, D. Newman, UCI machine learning repository (2007). URL: ⟨http://www.ics.uci.edu/ ~ mlearn/MLRepository.html⟩.
[6] T. Back, H. Hoffmeister, H. Schwefel, A survey of evolution strategies, in: Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 2–9.
[7] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, 1995.
[8] H.A. Bourlard, N. Morgan, Connectionist Speech Recognition: A Hybrid Approach, Kluwer Academic Publishers, Dordrecht, 1993.
[9] X. Cai, N. Zhang, G.k. Venayagamoorthy, D.C. WunschII, Time series prediction with recurrent neural networks trained by a hybrid pso-ea algorithm, Neurocomputing 70 (13–15) (2007) 2342–2353.
[10] E. Cantú-Paz, C. Kamath, An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems, IEEE Transactions on Systems, Man, and Cybernetics, Part B 35 (5) (2005) 915–927.
[11] Y. Chen, A. Abraham, B. Yang, Hybrid flexible neural tree based intrusion detection systems, International Journal of Intelligent Systems 22 (2007) 1–16.

[12] Y. Chen, A. Abraham, B. Yang, Feature selection and classification using flexible neural tree, Neurocomputing 70 (1-3) (2006) 305–313.
[13] Y. Chen, B. Yang, A. Abraham, Flexible neural trees ensemble for stock index modeling, Neurocomputing 70 (4-6) (2007) 697–703.
[14] Y. Chen, B. Yang, J. Dong, A. Abraham, Time series forecasting using flexible neural tree model, Information Sciences 174 (3–4) (2005) 219–235.
[15] C.A.C. Coelho, G.B.L. Lamont, D.A.V. Veldhuizen, Evolutionary Algorithms for Solving Multi-objective Problems (Genetic and Evolutionary Computation), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[16] P. Cortez, M. Rocha, J. Neves, Evolving time series forecasting arma models, Journal of Heuristics 10 (4) (2004) 415–429.
[17] M. Dirst, A.S. Weigend, Time Series Prediction: Forecasting the Future and Understanding the Past, Addison-Wesley, Reading, MA, 1994.
[18] M. Dorigo, G.D. Caro, Ant colony optimization: a new meta-heuristic, Proceedings of the Congress on Evolutionary Computation (1999) 1470–1477.
[19] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer, Berlin, 2003.
[20] F. Emmert-Streib, Influence of the neural network topology on the learning dynamics, Neurocomputing 69 (10–12) (2006) 1179–1182.
[21] K.P. Ferentinos, Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms, Neural Networks 18 (7) (2005) 934–950.
[22] R. Fletcher, Practical Methods of Optimization, second ed., Wiley-Interscience, New York, NY, USA, 1987.
[23] N. García-Pedrajas, C. Fyfe, Construction of classifier ensembles by means of artificial immune systems, Journal of Heuristics 14 (3) (2008) 285–310.
[24] N. García-Pedrajas, C. Hervás-Martínez, D. Ortiz-Boyer, Cooperative coevolution of artificial neural network ensembles for pattern classification, IEEE Transactions on Evolutionary Computation 9 (3) (2005) 271–302.
[25] N. García-Pedrajas, D. Ortiz-Boyer, C. Hervás-Martínez, Cooperative coevolution of generalized multi-layer perceptrons, Neurocomputing 56 (2004) 257–283.
[26] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Longman Publishing Co., Reading, MA, New York, 1989.
[27] S. Goonatilake, S. Khebbal (Eds.), Intelligent Hybrid Systems, Wiley, New York, NY, USA, 1994.
[28] T. Hanne, Global multiobjective optimization using evolutionary algorithms, Journal of Heuristics 6 (3) (2000) 347–360.
[29] S. Haykin, Neural Networks: A Comprehensive Foundation, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.
[30] H.B. Hwarng, H.T. Ang, A simple neural network for ARMA(p,q) time series, Omega 29 (4) (2001) 319–333.
[31] M.M. Islam, K. Murase, A new algorithm to design compact two-hidden-layer artificial neural networks, Neural Networks 14 (9) (2001) 1265–1278.
[32] N.K. Kasabov, Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering, MIT Press, Cambridge, MA, USA, 1996.
[33] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the International Joint Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[34] J.P.C. Kleijnen, S.M. Sanchez, T.W. Lucas, T.M. Cioppa, State-of-the-art review: a user's guide to the brave new world of designing simulation experiments, INFORMS Journal on Computing 17 (3) (2005) 263–289.
[35] M. Kordos, W. Duch, A survey of factors influencing mlp error surface, Control and Cybernetics 33 (2004) 611–631.
[36] K. Levenberg, A method for the solution of certain non-linear problems in least squares, Quarterly Applied Mathematics 2 (1944) 164–168.
[37] L.B. Liu, Y.J. Wang, D. Huang, Designing neural networks using pso-based memetic algorithm, in: Proceedings of the Fourth International Symposium on Neural Networks (ISNN'07), 2007, pp. 219–224.
[38] L. Ma, K. Khorasani, New training strategies for constructive neural networks with application to regression problems, Neural Networks 17 (4) (2004) 589–609.
[39] M. Mandischer, A comparison of evolution strategies and back-propagation for neural network training, Neurocomputing 42 (1) (2002) 87–117.
[40] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, SIAM Journal on Applied Mathematics 11 (2) (1963) 431–441.
[41] T. Masters, Signal and Image Processing with Neural Networks: a C++ Sourcebook, Wiley, New York, 1994.
[42] L.R. Medsker, Hybrid Intelligent Systems, Kluwer Academic Publishers, Norwell, MA, USA, 1995.
[43] B.L. Miller, D.E. Goldberg, Genetic algorithms, tournament selection, and the effects of noise, Complex Systems 9 (1995) 193–212.
[44] M.F. Møller, A scaled conjugate gradient algorithm for fast supervised learning, Neural Networks 6 (4) (1993) 525–533.
[45] P. Moscato, On evolution search, optimization, genetic algorithms and martial arts: towards memetic algorithms, Technical Report C3P 826, California Institute of Technology, Pasadena, CA, 1989.
[46] D.E. Rumelhart, J.L. McClelland, Explorations in the microstructure of cognition, in: Parallel Distributed Processing: Foundations, vol. 1, MIT Press, Cambridge, MA, USA, 1986.
[47] D.E. Rumelhart, P. Smolensky, J.L. McClelland, G.E. Hinton, Schemata and sequential thought processes in pdp models, in: Parallel Distributed Processing: Psychological and Biological Models, vol. 2, MIT Press, Cambridge, MA, 1986, pp. 7–57.
[48] J.J. Torres, M.A. Mu noz, J. Marro, P.L. Garrido, Influence of topology on the performance of a neural network, Neurocomputing 58–60 (2004) 229–234.
[49] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE 87 (9) (1999) 1423–1447.
[50] J. Yu, S. Wang, L. Xi, Letters: evolving artificial neural networks using an improved pso and dpso, Neurocomputing 71 (4–6) (2008) 1054–1060.
[51] J.P.T. Yusiong, P.C. Naval Jr., Training neural networks using multiobjective particle swarm optimization, Lecture Notes in Computer Science—Advances in Natural Computation vol. 4221 (2006) 879–888.

Leandro Maciel Almeida received the B.S. degree in Information Systems in 2004 from Lutheran University of Brazil/University Lutheran Centre of Palmas, Palmas, Tocantins, Brazil. He received the M.S. degree in Computer Science in 2007 from Federal University of Pernambuco, Brazil. Currently, he is a Ph.D. student of the Centre of Informatics at Federal University of Pernambuco. His research interests include artificial neural networks, pattern recognition, hybrid intelligent systems and evolutionary computation.

Teresa Bernarda Ludermir received the Ph.D. degree in Artificial Neural Networks in 1990 from Imperial College, University of London, UK. From 1991 to 1992, she was a Lecturer at Kings College London. She joined the Center of Informatics at Federal University of Pernambuco, Brazil, in September 1992, where she is currently a Professor and the Head of the Computational Intelligence Group. She has published over a 150 articles in scientific journals and conferences, three books in NN and organized two of the Brazilian Symposium on Neural Networks. She is one of the editors-in-Chief of the International Journal of Computation Intelligence and Applications. Her research interests include weightless NN, hybrid neural systems and applications of NNs.