Neural Comput & Applic
DOI 10.1007/s00521-010-0504-3

ORIGINAL ARTICLE

Metaheuristics for the feedforward artificial neural network (ANN) architecture optimization problem

Adenilson R. Carvalho · Fernando M. Ramos · Antonio A. Chaves

Received: 21 September 2009 / Accepted: 22 November 2010

Abstract This article deals with evolutionary artificial neural network (ANN) and aims to propose a systematic and automated way to find out a proper network architecture. To this, we adapt four metaheuristics to resolve the problem posed by the pursuit of optimum feedforward ANN architecture and introduced a new criteria to measure the ANN performance based on combination of training and generalization error. Also, it is proposed a new method for estimating the computational complexity of the ANN architecture based on the number of neurons and epochs needed to train the network. We implemented this approach in software and tested it for the problem of identification and estimation of pollution sources and for three separate benchmark data sets from UCI repository. The results show the proposed computational approach gives better performance than a human specialist, while offering many advantages over similar approaches found in the literature.

Keywords Neural network architecture · Optimization algorithms · Evolutionary computation

A. R. Carvalho (✉) · F. M. Ramos · A. A. Chaves
Laboratory for Computing and Applied Mathematics (LAC), Brazilian National Institute for Space Research (INPE), São José dos Campos, São Paulo, SP, Brazil
e-mail: adenilson@lac.inpe.br

F. M. Ramos
e-mail: fernando@lac.inpe.br

A. A. Chaves
e-mail: chaves@lac.inpe.br

1 Introduction

Artificial neural networks (ANNs) have been a topic of interest in research institutes worldwide. Although much has been proposed and studied about, there are still many issues about the model of ANNs that need to be better understood. There are no prior assurances the adopted model will be successful in the task for which it was designed. Much has been studied and researched in try to find systematic ways to adjust an ANN model for a problem in particular aiming at optimizing its overall performance. These surveys are based on almost all aspects in the ANNs modeling such as the different types of activation function, the weights initialization, the training data collection, pre- and post-processing, the training algorithms, and error functions. However, theoretical and empirical evidences found in literature show the search and definition of an ANN optimal or near-optimal architecture are the most important factors for model computational performance, efficiency, and accuracy [1].

We can formulate the problem of identifying an optimized network architecture as a search in the space of neural network topologies where each point represents a possible architecture. If we associate with each point (architecture) a performance level based on some optimality criterion (such as complexity, minimum square error, and learning speed), we can build and display a surface in the three-dimensional space. Finding an optimal architecture in this case would be equivalent to find the highest or the lowest point of this surface, that is, its global extreme points depending on the criteria used [2].

In this work, we apply a group of metaheuristics to solve this problem. There are no registers in the literature of any attempt toward use the GEO and the VNS as an optimization method for the best network architecture problem. Further, we define the computational complexity of a feedforward ANN architecture as a function of the total number of weights and bias present in its structures and the time required for the network learning. We derive from this a penalty term that is used to evaluate the objective function to avoid overcomplex networks architectures. The use combined of these elements is also a contribution of this work.

Another contribution of this work was to include two weighting factors (ρ_1 and ρ_2, see Sect. 3.3), that measure the influence of the generalization and training errors in the algorithmic search process, allowing to us to fine tune the progress of the optimization.

1.1 The feedforward network

Considering the universe of possible ANNs, the feedforward neural network was the first and arguably simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. By a feedforward ANN architecture, we mean a number of constitutive elements among which stands out the number of layers, the number of processing units (neurons) in each layer, the activation function, and the training algorithm (because it determines the final value of weights and bias). Beyond these, there are two other factors also important to define an extended ANN architecture: the learning rate (α) and the momentum term (η). These two terms play an important role in the BP because they control the convergence speed of the ANN. It has been noted that by using this parameters effectively reduces the network training time and accelerates the convergence to the optimum [3].

The learning rate is a term used in the ANN learning rule to update the connection weights. Its value is restricted to the interval [0, 1] and controls the amount that a connection weight changes in response to an error signal. If the selected learning rate is very large, then the local minimum can be overcome constantly, resulting in slow oscillations and convergence to a minor error state. If the learning rate is low, the number of iterations needed can be very large, resulting in low performance [4].

The momentum term is a commonly used to update rule introduced by [3] which is a fraction of the previous weight change to the current weight change during the weights adjustment stage. Adding such term can help smooth out the descent path by preventing extreme changes in the gradient due the local anomalies.

1.2 The multilayer perceptron

This paper proposes a method for finding out the best ANN architecture by using multilayer perceptron feedforward networks with backpropagation training algorithm. A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto an appropriate output set. It is a variation of the standard linear perceptron in which it uses three or more neuron layers (nodes) with nonlinear activation functions. It is more powerful than the perceptron because it can distinguish data that are not linearly separable, or separable by a hyperplane [5].

In addition, we use the backpropagation (BP) algorithm, that is, one of the most popular methods for training multilayer feedforward neural networks. Essentially, the BP algorithm is a supervised learning method, which provides a way to calculate the gradient of the error function efficiently. Multilayer perceptrons using a backpropagation algorithm are the standard algorithm for any supervised learning pattern recognition process and the subject of current research in computational neuroscience and parallel distributed processing. They are useful in research by their ability to solve problems stochastically, which often allows one to get approximate solutions for complex problems.

The rest of this paper is organized as follows. Section 2 addresses Evolutionary computation. In the Sect. 3, we explain in detail the methodology adopted in this work. The obtained results and discussions based on the performance comparison with two other approaches to the same problem are presented in the Sect. 6 Finally, in the Sect. 7, we draw some conclusions and future works.

2 Evolutionary computation

There is no a clear and unambiguous indication of how we find the best architecture among the many choices presented. Further, we not even know in advance the correct network topology to be applied. In practice, this problem is usually solved in part by using empirical methods based on repetitive trial and error method. In this case, we could say the success of this method depends almost solely and exclusively of previous experience and intuition of expert human behind the scene, involving therefore a high degree of uncertainty. In such methods, we spent much CPU time in unnecessary simulations and that not cover the entire available architecture choices set for a particular problem.

Also, we almost always found a suboptimal solution that corresponds to a lower solution if compared to that obtained by using an autonomous search method.

It is therefore an exhausting task to design and build an ANN for a specific problem. There are a few rigorous project principles that are available to define the ANN and, however, many parameters to adjust. The constructive methods are an alternative to artificial neural networks designing, but the computational cost required for certain applications can be prohibitive, besides the difficulty presented in the high-dimension input spaces treatment. Therefore, the goal here is to consider advanced evolutionary computation techniques, which also promote achieving ANN architectures that are dedicated, specific to each problem.

The evolutionary computation is a branch of computer science that proposes an alternative paradigm to conventional data processing. This new paradigm, unlike the conventional, does not require, to solve a problem, the prior knowledge of a way to find a solution. It is based on evolutionary mechanisms found in nature, such as self-organization and adaptive behavior (see [6, 7]). These mechanisms have been discovered and formalized by Darwin in his theory of natural evolution, according to which, life on earth is the result of a process where the most suitable and adapted having therefore more chances of reproducing are selected by the environment.

We can divide the evolutionary computing into two main groups: evolutionary computing to train ANN or to adjust its weights and evolutionary computing for ANN architectures definition [8].

In a similar way, [9] identified two approaches to code the ANN architecture in a binary string or in a decimal representation. One is the strong specification scheme (or direct encoding scheme) where a network's architecture is explicitly coded. The other is a weak specification scheme (or indirect encoding scheme) where the exact connectivity pattern is not explicitly represented.

We can include as examples of the application of the strong specification scheme [9–13]. On other hand, the applications of the weak specification scheme include [1, 2, 14, 15, 16, 17]. An extensive comparison between various methods of coding can be found at [18].

Our approach falls into this last category since it uses an indirect encoding scheme for finding out the best architecture to solve a problem in particular. However, unlike most other approaches, besides employing a purely evolutionary strategy like genetic algorithm (GA) or ant colony optimization (ACO) for example, we also applied some alternative combinatorial methods like Simulate Annealing or Variable Neighborhood Search that were very successful in solving problems with too many parameters that exhibit a clear exponential trend in the number of feasible solutions.

3 Methodology

This paper presents an innovative approach for optimizing feedforward MLP ANN architecture based on the use of global search metaheuristics.

To solve the optimal ANN architecture search problem, we implement the algorithm known as Generalized Extremal Optimization (GEO) [19–21], the Variable Neighborhood Search (VNS) [22, 23], Simulated Annealing (SA) [24], and canonical genetic algorithm (GA-based version [25, 26].

3.1 Candidate solution representation

Optimal network architectures can be evolved to fit a given task at hand. Both the network topology representation and search operators used in our experiments are the most important issues to considerer in the evolution of architectures.

In our approach, we applied the above-mentioned optimization algorithms to evolve: (1) the number of hidden layers;(2) the number of processing elements present in each hidden layer;(3) the ANN learning rate;(4) the momentum term; and (5) the ANN activation function.

The learning rate (α) can take any of the values given by the sum below:

$$\alpha = \sum_{i=1}^{k} a_k \times 2^{-k} \tag{1}$$

where $a_k \in \{0,1\}$ and $k \in N$. A similar expression is adopted for the momentum term (η). In this work, we use $k = 6$ for α and $k = 4$ for η.

The possible values for the remaining parameters are shown in the Table 1 below:

The ANN built from information encoded in binary structures is trained on simulated data generated from the LAMBDA model (see Sect. 4), which is a well-known forward model and that is widely used in dispersion pollution studies.

For the simulation performed in this paper, we code the parameters of the problem in a 19-bit binary string. The first and second groups of six and four bits correspond to the learning rate and momentum, respectively. We represent the activation function type employed in the network training and the number of hidden layers permitted in the network solution by the third and fourth groups consisting

Table 1 Interval of values assumed by the parameters that define a network architecture

Parameter	Values
Activation function	{Tanh[a], sigmoid, log[b], gaussian}
Hidden layers	{1, 2, 3}
Neurons in each hidden layer	{1, 2, …, 32}

[a] Hiperbolic tangent

[b] Logarithmic

of two bits each one that follows. The last group of five bits is related to the number of neurons in each layer. Because of the binary encoding scheme and the number of bits used, the number of neurons in each hidden layer is restricted to [0, 31] since $1 + 2 + 4 + 8 + 16 = 31$. To further clarify the applied encoding scheme, two different candidate solutions, and its decoded forms, are provided in Fig. 1.

When we decrease or increase the string of bits, we are expanding or reducing the number of possible architectures (the architecture space dimension), thus ensuring the generality of the methodology.

It is easy to show the number of possible solutions (ANN architectures) represented by binary strings follows a simple power law given by the equation $n = 2^b$ where b is the number of bits in the binary string solution and n the number of solutions. So, in this work, our search space was formed by 524288 possible network architectures solutions that matches to 2^{19} binary combinations.

In this work, we adopt the limit of up to two hidden layers and 32 neurons in each layer only for practical reasons (computational time). Anyhow, studies have shown that for feedforward ANNs with continuous nonlinear activation function, a hidden layer with an arbitrarily large number of neurons is enough because of their universal approximation property [27–29]. So, the employed limit is more than enough for most applications.

3.2 Network learning and test

After coding each candidate solution into a network architecture, we perform the network training. In the next step, the final model is tested with the test set data, which is the model generalization test, to ensure that the results on the selection and training set are real, and not artifacts of the training process. Of course, to fulfill this role properly,

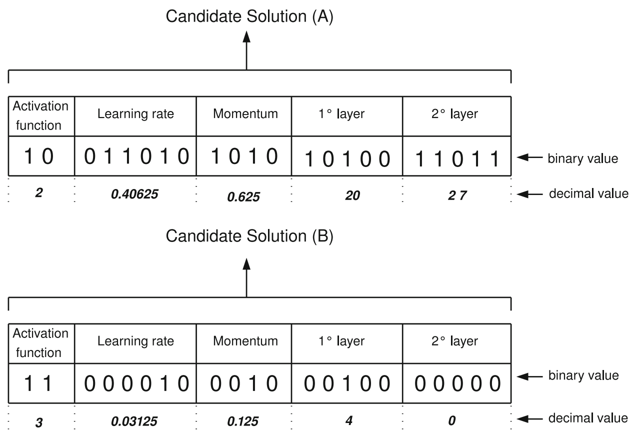


Fig. 1 Binary encoding examples of two candidate solutions to network architectures and its decimal representation

the test set is used only once and contains data that do not belong to the training set.

At this stage, the ANN has "learned" to identify the gas emissions of an area source or accurately classify a given input pattern and is also capable of predicting the untrained output patterns.

After that we assign a score to the solution formed according to an objective function (see Sect. 3.3 below) previously defined. So, we build a new candidate solution using the same method described above and the process repeats itself.

It is important to mention that in this work, we used a random weight initialization function for all weights and bias with values randomly distributed, chosen from interval $[-1, 1]$. We also employed the early stopping training strategy using cross validation for the network training as described in [28].

The ANN model optimization that aims to obtain an optimal or suboptimal network architecture follows similar steps no matter what the metaheuristics employed. These common steps for search algorithms are described below.

3.3 Objective function applied

In this work, we deal with a complex combinatorial optimization problem where we are interested in finding a minimum cost network topology, showing better performance with less computational overload possible. To that end, we built an objective function that consists of a combination of two criteria for error and one penalty that can be expressed by:

$$f_{\text{obj}} = \text{penalty} \times \left(\frac{\rho_1 \times E_{\text{train}} + \rho_2 \times E_{\text{gen}}}{\rho_1 + \rho_2} \right) \tag{2}$$

where ρ_1 and ρ_2 ($\rho_1, \rho_2 > 0$; $\rho_1, \rho_2 \in \Re$) are adjustment factors used to weigh the degree of importance attributed to the training and generalization errors, respectively. Usually we have $\rho_1 \leq \rho_2$ because we are interested in models that provide good answers when confronted with an input data set that does not belong to training set, that is, that is known to generalize. In particular, for this work, we use three sets of values for ρ_1 and ρ_2 (see Table 2).

The function f_{obj} is the function whose value should be minimized by algorithms implemented. Thus, it becomes clear that f_{obj} is composed by the sum of errors in the training and generalization stages multiplied by the penalty due to the architecture complexity of the ANN in question. The minimum value of f_{obj} corresponds to a simple architecture and exhibits a consistent behavior in the space of solutions combined with low training and generalization errors. Here, we mean by "simple" a solution whose architecture is reduced in terms of weight numbers and learning time.

Table 2 Values applied in each one of the experiments performed in this work for the weighting factors applied to the training (ρ_1) and generalization (ρ_2) errors

Experiment	ρ_1	ρ_2
#1	1.0	0.1
#2	1.0	1.0
#3	0.1	1.0

3.3.1 Detailing the criteria used to compose the objective function

Training error Criteria This term is important because it provides us with a quantitative indication of the network training level and is directly related to the ability to memorize the network. The mean square error formula used to calculate the training error can be expressed as follows:

$$E_{\text{train}} = \sqrt{\frac{1}{2N} \times \sum_{i=1}^{N} \sum_{j=1}^{M} \left[Y_{m_{ij}} - Y_{d_{ij}}\right]^2} \tag{3}$$

where E_{train} is the training error, $Y_{d_{ij}}$ is the desired output value (target) in *i*th training pattern for the *j*th ANN output layer neuron, $Y_{d_{ij}}$ is the ANN response in *i*th training pattern for the *j*th ANN output layer neuron, N is the number of training patterns or examples presented to the network, and M is the number of neurons in output layer.

Generalization error criteria This refers to the ability of ANN to identify and respond to patterns that are similar but not identical to the patterns with which the network was trained. It appears more important than the training error as a parameter signaling the performance of the ANN model in most applications. The formula used to set the error is as follows:

$$E_{\text{gen}} = \sqrt{\frac{\sum_{i=1}^{M} \left[Y_{m_i} - Y_{d_i}\right]^2}{M}} \tag{4}$$

where E_{gen} gen is the generalization error, Y_{d_i} is the target value for the *i*th output neuron, Y_{m_i} is the ANN predicted value of the *i*th output neuron, and M is the number of neurons in output layer.

Penalty due the network complexity The third criterion determines the influence of the ANN architecture on the objective function values. We can define the computational complexity of a feedforward ANN architecture as the total number of weights and bias present in its structure. When we talk about optimization of an ANN architecture, we are interested in obtaining models or architectures that present a performance as close as possible to a global optimum for the problem in question or, at least, that produce suboptimal answers better than the answers provided by a non-optimized model. In other words, we look for network architectures with low training and generalization errors. To this end, we developed a criterion that favors lighter architectures by applying a penalty term to the objective function. By using smaller architectures, we avoid the network training data overadjustment, known as "overfitting" [5], and speed up the training process because there will be fewer processing units and weight factors to be calculated.

In an optimization problem, such penalty is incorporated to the objective function in such a way to constraint the universe of possible ANN architectures. It penalizes larger and complex ANN architectures with too many neurons in the hidden layers or that need too much CPU time (learning time). Ideally, the penalty value should be an exponential function of the ANN complexity and not a linear function as we might expect. This occurs because the sum of the weights and bias number also increases in a nonlinear way with the number of hidden layers and hidden neurons. Thus, the penalty applied would be gradually increasing as the total number of weights and bias is low and rapidly when the architecture is complex.

So the general shape of the criterion for an ANN architecture penalty would be given by

$$P_1 = a \times e^{f(x)} \tag{5}$$

where a is a constant and $f(x)$ a function of the weights and bias total number, usually denoted by an exponential function.

We also included restrictions on the learning time of the patterns present in the network input. To that end, we introduced a second penalty term that considers the number of cycles or epochs needed to perform the training. In this work, we define this factor penalty as a linear function in the number of epochs, that is, $g(y)$, that has the form $P_2 = b \times y + c$.

where b is a constant that measures the slope of the line and y represents the number of training epochs or cycles.

So, grouping the terms defined above, we have the following global expression for the penalty applied to the model:

$$Penalty = P_1 + P_2 \tag{6}$$

In this work, specifically, we assume the following global expression for the penalty due to the model complexity:

$$\text{Penalty} = \underbrace{5 \times 10^{-8} \times e^{x^2}}_{P_1} + \underbrace{5 \times 10^{-5} \times y + 1}_{P_2} \tag{7}$$

where x is the number of weights in the connections and y the number of epochs necessary in the network training.

Both P_1 as P_2 can be represented by any continuous monotonic function and strictly increasing in the range $[1, \infty]$.

Figure 2 below displays a three dimensional for the penalty function applied in this work. As previously mentioned, in applying such a penalty function to our problem, we prefer solutions with fewer neurons in each hidden layer and that, at the same time, not need many iterations to converge on an appropriate solution.

4 First case study: identification and estimation of atmospheric pollutant sources

In this work, we used a multilayer perceptron ANN to solve the problem of estimating air pollution sources (see [30]).

We tested the methodology using data from the dispersal Lagrangian stochastic LAMBDA model for pollution dispersion (acronym for LAgrangean Model for Buoyant Dispersion in Atmosphere), developed to study the pollutants transport and diffusion on land plan (see [31, 32]).

The meteorological data used by LAMBDA model to simulate the dispersion of particles by the wind are extracted from Copenhagen [30] and displayed in the Table 3. This table shows the speeds and average directions of three strata of wind obtained in five different hours and correspond to the measurements performed on 19/10/1978 (Grining).

The data needed for the problem solving were provided from six sensors, with a detection area of 0.1 m × 0.1 m, positioned at a height of 10 m and installed in the region with the layout defined by Table 4.

So, the forward model described in [33] is used to produce 500 data sets that are used by the inversion model. The inverse problem to be solved by the ANN model and that can be found in [34] consisted of obtaining the pollutants concentration in the six sensors from sources properly discretized in 12 subareas according to the Fig. 3.

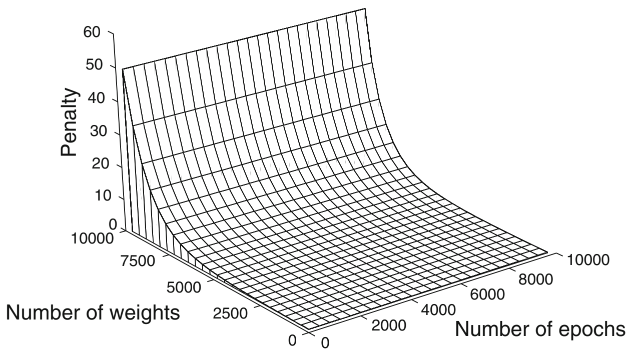


Fig. 2 Three-dimensional visualization tridimensional of the penalty function applied to the problem

Table 3 Meteorological data for average wind speed and direction extracted from the Copenhagen experiment

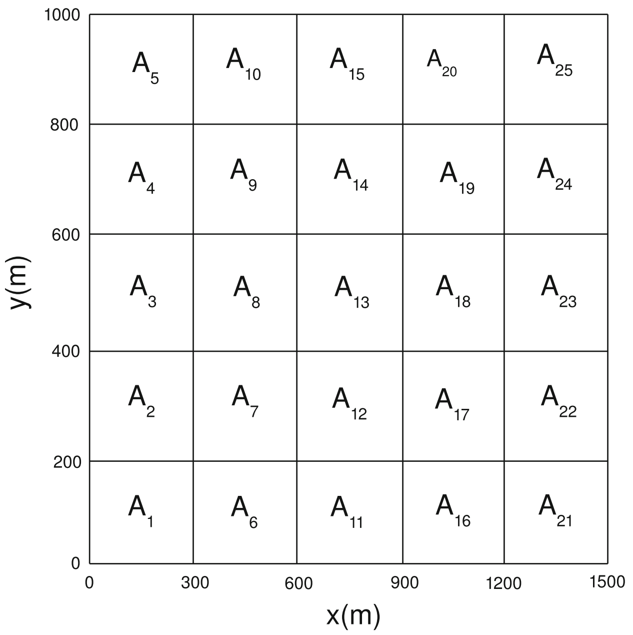
Hour	Speed $\overrightarrow{V}$ (m/s)			Direction $\overrightarrow{U}$ (°)		
12:05	2.6	5.7	5.7	290	310	310
12:15	2.6	5.1	5.7	300	310	310
12:25	2.1	4.6	5.1	280	310	320
12:35	2.1	4.6	5.1	280	310	320
12:45	2.6	5.1	5.7	290	310	310

Source: [33]

Table 4 Position of the sensors in the study area

Sensor	Position x (m)	Position y (m)
1	400	500
2	600	300
3	800	700
4	1,000	500
5	1,200	300
6	1,400	700

Source: [33]



A_5 A_{10} A_{15} A_{20} A_{25}
A_4 A_9 A_{14} A_{19} A_{24}
A_3 A_8 A_{13} A_{18} A_{23}
A_2 A_7 A_{12} A_{17} A_{22}
A_1 A_6 A_{11} A_{16} A_{21}
y(m)
1000
800
600
400
200
0
0
300
600
900
1200
1500
x(m)

Fig. 3 Discretization of the study area into a rectangular grid (5 × 5) of cells. Source: [33]

The visual representation of the sensors can be seen in the Fig. 4.

5 Benchmark data sets

Additionally, a number of experiments were conducted with standard benchmark data sets of the University of

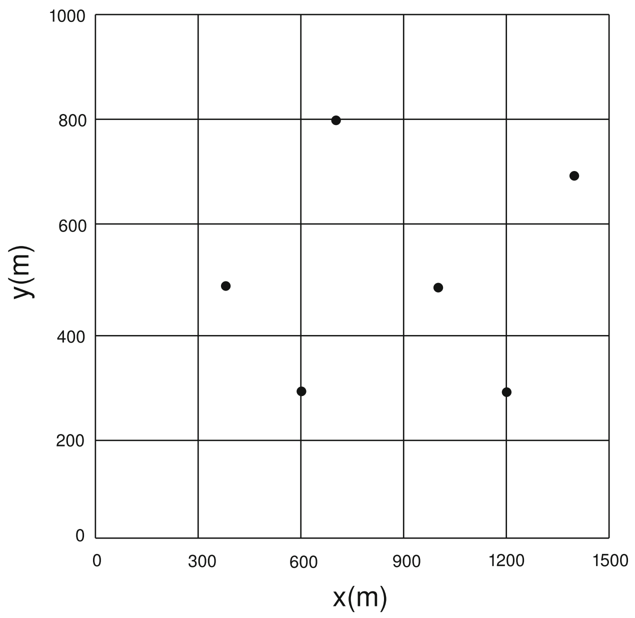


Fig. 4 Sensor positioning, represented by *filled circle*, in the study area. Source: [33]

California Irvine (UCI) machine learning repository [35] to test the performance of our system. For this study, we used the Wisconsin Breast Cancer Data Set, the adult data set, and Glass Identification data set. The first one, The Wisconsin Breast Cancer Data set, consists of 569 cases, of which 357 are diagnosed as benign and the remaining 212 are known to be malignant. There are no missing attributes in the data set, and in this case, we are interested in classifying the breast tumor as benign and malignant. The adult data set contains the questionnaire data of the "adult" database (originally called the "Census Income" Database) formed by a data frame with 48,842 observations on the following 15 variables. It was originally used to predict whether income exceeds USD 50 K/year based on census data. Finally, the third one is greatly imbalanced and extremely sparse, having seven classes with only a few instances in each class. It is composed of 214 instances without missing values of six different classes and is an example of a data set with a special overlap problem. The data set consists of nine inputs, which correspond to six different classes of glass i.e. three types of window, containers, and headlamps. The three window classes were extremely overlapping and difficult to separate.

6 Computational experiments

The following are the computational results obtained from experiments performed for each metaheuristic used in such work. All algorithms are implemented in the Java language and computer tests were conducted under the Linux operating system, in a microcomputer processor with AMD Athlon (tm) 64 Processor 3,200+, 1.53 GHz and 1 Gb MB of RAM.

6.1 Results for the identification and estimation of atmospheric pollutant sources

Initially, three computational experiments were conducted in accordance with Table 2 (see Sect. 3.3). The parameters matching to the best found solution after a predetermined number of consecutive objective function evaluations are displayed in the Tables 5, 6, and 7.

In a first experiment, we assigned a larger weight to the ability of ANN to learn the presented patterns at the expense of its generalization ability. We present the results in Table 5. In the second experiment, the ANN learning and generalization errors had equal weights in the objective function arrangement. The results of these simulations are shown in the Table 6 and finally, in the Table 7, we simulate a more realistic scenario, in which the ability to generalize, or find a correct output value for an unknown ANN pattern, it is more relevant than the ability to learn the model.

The analysis of this results highlights the virtues of using an automated evolutionary approach in the search for optimal parameters of ANN-based models.

Thanks to approach employed, we were able to produce ANN architectures with great generalization power and with low computational cost for the problem considered. We can note this in the experiments $\{6, 12\}$ and $\{3, 12\}$ of Tables 6 and 7, respectively.

The best network architecture found was obtained after 5,000 iterations of the canonical genetic algorithm and corresponds to the network parameters configuration shown in the experiment #12 of Table 6. The solution leads to a network architecture with two hidden layers and few neurons in each of them (5 and 7, respectively) and suggests to apply a hyperbolic tangent activation function in the intermediate layers. Curiously, this solution was obtained giving equal weight to both network training and generalization errors. Other solutions with good tradeoff between computational cost required and performance offered were obtained in experiment 12 of Table 7 and experiment 6 of Table 6.

Moreover, when comparing the tables above, it appears the weighting assigned to the training and generalization errors has influence on the final solution of architecture obtained by the search algorithm. As we might expect, the values for the generalization errors shown in Table 5 are larger on average than those of Table 7 and and this difference becomes greater as we increase the number of epochs for training the network. This is due the fact we credited more weight to the training error at the expense of

Table 5 Best ANN architectures encountered by VNS, SA, GEO, and GA algorithms

Exp	Metaheuristics	# Evaluations F_{obj}	Architecture	α	η	$\phi(\cdot)$	E_{trein}	E_{gen}
#1	VNS	100	$6 \times 31 \times 12 \times 12$	0.34375	0.1875	Sigmoid	0.01502	0.5704
#2	VNS	1,000	$6 \times 5 \times 4 \times 12$	0.25	0.375	Sigmoid	0.02673	0.1924
#3	VNS	5,000	$6 \times 6 \times 15 \times 12$	0.25	0.1875	Tanh	0.00913	0.1429
#4	SA	100	$6 \times 6 \times 11 \times 12$	0.265625	0.125	Logarithmic	0.02054	0.5338
#5	SA	1,000	$6 \times 15 \times 14 \times 12$	0.75	0.5	Sigmoid	0.01518	0.2887
#6	SA	5,000	$6 \times 10 \times 11 \times 12$	0.296875	0.0625	Tanh	0.00706	0.1478
#7	GEO_{var}	100	$6 \times 10 \times 7 \times 12$	0.546875	0.0625	Sigmoid	0.01906	0.4367
#8	GEO_{var}	1,000	$6 \times 26 \times 12$	0.28125	0.1875	Tanh	0.02034	0.7717
#9	GEO_{var}	5,000	$6 \times 5 \times 6 \times 12$	0.203125	0.0	Tanh	0.0113	0.2841
#10	GA	100	$6 \times 21 \times 12$	0.046875	0.8125	Sigmoid	0.03128	0.6356
#11	GA	1,000	$6 \times 6 \times 11 \times 12$	0.140625	0.375	Sigmoid	0.02177	0.4527
#12	GA	5,000	$6 \times 5 \times 16 \times 12$	0.015625	0.5	Tanh	0.00962	0.1608

In such experiments, the weighting factors given to training (ρ_1) and generalization (ρ_2) errors were 1.0 and 0.1, respectively

Table 6 Best ANN architectures encountered by VNS, SA, GEO, and GA algorithms

Exp	Metaheuristics	# Evaluations F_{obj}	Architecture	α	η	$\phi(\cdot)$	E_{trein}	E_{gen}
#1	VNS	100	$6 \times 15 \times 16 \times 12$	0.125	0.1875	Gauss	0.02992	0.5030
#2	VNS	1,000	$6 \times 11 \times 30 \times 12$	0.578125	0.4375	Sigmoid	0.01842	0.4120
#3	VNS	5,000	$6 \times 8 \times 8 \times 12$	0.046875	0.75	Tanh	0.02333	0.1651
#4	SA	100	$6 \times 8 \times 10 \times 12$	0.984375	0.5	Sigmoid	0.02580	0.3358
#5	SA	1,000	$6 \times 14 \times 15 \times 12$	0.109375	0.625	Tanh	0.00739	0.1527
#6	SA	5,000	$6 \times 5 \times 6 \times 12$	0.140625	0.5	Tanh	0.01676	0.0756
#7	GEO_{var}	100	$6 \times 3 \times 16 \times 12$	0.09375	0.25	Tanh	0.02215	0.8788
#8	GEO_{var}	1,000	$6 \times 7 \times 18 \times 12$	0.078125	0.375	Logarithmic	0.02501	0.4381
#9	GEO_{var}	5,000	$6 \times 23 \times 12$	0.1875	0.625	Sigmoid	0.02082	0.6103
#10	GA	100	$6 \times 27 \times 12$	0.984375	0.375	Tanh	0.02620	0.7358
#11	GA	1,000	$6 \times 17 \times 10 \times 12$	0.03125	0.6875	Gauss	0.03079	0.4407
#12	GA	5,000	$6 \times 5 \times 7 \times 12$	0.109375	0.4375	Tanh	0.01589	0.0716

In such experiments, the weighting factors given to training (ρ_1) and generalization (ρ_2) errors were equal to 1.0

the network generalization ability in the objective function composition in the experiments of the Table 5.

By comparison, it can be inferred from results the topology (architecture) produced by GEO_{var} was lower than those produced by other metaheuristics. SA, GA, and VNS yielded solutions with few hidden nodes at a low computational cost. The ratio between the best and the worst solution obtained in these experiments, considering the same number of objective functions evaluations in each case, was 0.2214, 0.1978, and 0.1173 for 100, 1,000, and 5,000 evaluations, respectively. These values reveal another interesting fact about the proposed model evolutive dynamic: if we increase the number of objective function evaluations, some algorithms or metaheuristics tend to excel the others.

The next step was to compare the results got with our strategy with other non-optimized approaches for the problem. In the Table 8, we compare the results of the optimization algorithms with the solution obtained by [33] and [34]. Luz et al. applied in their work two stochastic techniques: optimization by particle swarm (particle swarm optimization - PSO) and optimization by colony of ants (ant colony optimization - ACO). In a recent work, Paes et al. used an MLP neural network with simple parameters adjustment, that is, without any concern about how to get a little more optimized network architecture for the problem.

The low error, both in absolute and relative terms, obtained by the optimization technique shows the superiority of the approach over other simpler strategies. In [33], we can see some results divergent and with many oscillations around the exact profile. Instead, if we consider the strategy employed by [34], what we can note is a clear tendency to underestimate the sources (areas A_{12}–A_{19}) and overestimate the sinks (areas A_2–A_9).

Table 7 Best ANN architectures encountered by VNS, SA, GEO, and GA algorithms

Exp	Metaheuristics	# Evaluations F_{obj}	Architecture	α	η	$\phi(\cdot)$	E_{trein}	E_{gen}
#1	VNS	100	$6 \times 18 \times 25 \times 12$	0.484375	0.625	Sigmoid	0.02083	0.4371
#2	VNS	1,000	$6 \times 9 \times 17 \times 12$	0.296875	0.125	Tanh	0.0167	0.1753
#3	VNS	5,000	$6 \times 17 \times 2 \times 12$	0.109375	0.625	Logarithmic	0.01548	0.1126
#4	SA	100	$6 \times 12 \times 12 \times 12$	0.21875	0.0	Tanh	0.01066	0.3004
#5	SA	1,000	$6 \times 15 \times 14 \times 12$	0.75	0.5	Sigmoid	0.01518	0.2887
#6	SA	5,000	$6 \times 3 \times 13 \times 12$	0.0625	0.5	Logarithmic	0.02228	0.1331
#7	GEO_{var}	100	$6 \times 25 \times 12$	0.453125	0.0	Tanh	0.02867	1.2825
#8	GEO_{var}	1,000	$6 \times 22 \times 15 \times 12$	0.046875	0.75	Logarithmic	0.01727	0.4073
#9	GEO_{var}	5,000	$6 \times 5 \times 30 \times 12$	0.28125	0.125	Tanh	0.01264	0.3503
#10	GA	100	$6 \times 23 \times 14 \times 12$	0.234375	0.125	Logarithmic	0.03048	1.3568
#11	GA	1,000	$6 \times 31 \times 2 \times 12$	0.015625	0.875	Logarithmic	0.01759	0.5527
#12	GA	5,000	$6 \times 6 \times 9 \times 12$	0.25	0.375	Logarithmic	0.01417	0.1195

In such experiments, the weighting factors given to training (ρ_1) and generalization (ρ_2) errors were 0.1 and 1.0, respectively

Table 8 Network performance comparison using noisy input data (5% white gaussian noise)

Area	Exact	(PSO[a]) [33]	ANN architecture $6 \times 30 \times 12$ [34]	Optimized ANN architecture $6 \times 5 \times 7 \times 12$
A_2	10	09.34	10.97	9.89
A_3	10	10.07	11.02	9.85
A_4	10	11.26	11.07	9.88
A_7	10	10.95	10.91	10.01
A_8	10	10.93	10.79	9.97
A_9	10	14.99	11.17	9.92
A_{12}	20	20.79	18.43	20.11
A_{13}	20	19.83	18.44	20.20
A_{14}	20	13.06	18.41	20.14
A_{17}	20	18.72	18.26	20.24
A_{18}	20	22.76	18.43	20.13
A_{19}	20	22.76	18.43	20.19

The values are in gram of pollutant per millimeter every tenth of a second

[a] Particle swarm optimization

6.2 Results for the benchmark data sets

Experiments were conducted with three UCI data sets by using the proposed optimization model. The results express the values for the network configuration parameters after 10,000 epochs. Here, similarly to the previous section, we considered three scenarios for the arrangement of the weighting factors ρ_1 and ρ_2. We shown this additional results obtained with the UCI data sets in Tables 9, 10 and 11 below.

The results for the UCI data set benchmark classification confirmed the trends identified in the first study case. VNS and GA metaheuristics were the best performing in the previous experiment and, similarly, also accounted for the highest marks here. For the glass data set, the canonical genetic algorithm (CGA) performed better in the classification of glass types with an accuracy of 63.86%, which can be considered a good match if compared with the results obtained by trial and error traditional approach for multilayer perceptron neural networks. VNS metaheuristic gets the best results for Wisconsin and adult data set classification by generating lightweight solutions presenting low computational complexity, although Simulate Annealing and GA have also been efficient in the task of generating good network topologies. Considering the data sets used in this work, the methodology was able to generate MLP topologies automatically, with much fewer connections than the maximum number allowed.

It is important to point out the focus of this work does not concern the use and posterior comparison of the metaheuristics employed, nor even to find out the best solution for the problem considered, but to propose new alternatives for the "blind search" strategy, commonly addressed by a trial and error iterative process. Obviously, the results obtained and displayed on Tables 9, 10 and 11 above could also be obtained by using other metaheuristics.

Table 9 Test set accuracy rate (%) results including best ANN architectures encountered by VNS, SA, GEO and GA algorithms

UCI data set	Metaheuristic	Test set accuracy rate (%)	Architecture	α	η	$\phi(\cdot)$
Wisconsin	VNS	97.50	$6 \times 16 \times 11 \times 12$	0.21875	0.1875	Gauss
Wisconsin	SA	96.63	$6 \times 31 \times 5 \times 12$	0.25	0.375	Gauss
Wisconsin	GEO_{var}	96.72	$6 \times 11 \times 3 \times 12$	0.140625	0.1875	Logarithmic
Wisconsin	GA	97.27	$6 \times 4 \times 6 \times 12$	0.26562	0.25	Logarithmic
Adult	VNS	83.82	$6 \times 22 \times 12$	0.78125	0.5	Sigmoid
Adult	SA	84.75	$6 \times 3 \times 14 \times 12$	0.046875	0.8125	Sigmoid
Adult	GEO_{var}	83.80	$6 \times 10 \times 7 \times 12$	0.09375	0.375	Tanh
Adult	GA	84.03	$6 \times 6 \times 8 \times 12$	0.140625	0.25	Tanh
Glass	VNS	58.92	$6 \times 15 \times 11 \times 12$	0.125	0.4375	Logarithmic
Glass	SA	57.88	$6 \times 17 \times 8 \times 12$	0.984375	0.5	Logarithmic
Glass	GEO_{var}	56.80	$6 \times 20 \times 12$	0.859375	0.0	Tanh
Glass	GA	61.83	$6 \times 3 \times 18 \times 12$	0.109375	0.25	Logarithmic

In such experiments the weighting factors given to training (ρ_1) and generalization (ρ_2) errors were 1.0 and 0.1, respectively

Table 10 Test set accuracy rate (%) results including best ANN architectures encountered by VNS, SA, GEO, and GA algorithms

UCI data set	Metaheuristic	Test set accuracy rate (%)	Architecture	α	η	$\phi(\cdot)$
Wisconsin	VNS	99.31	$6 \times 6 \times 13 \times 12$	0.109375	0.625	Gauss
Wisconsin	SA	97.98	$6 \times 5 \times 4 \times 12$	0.5625	0.0625	Logarithmic
Wisconsin	GEO_{var}	96.12	$6 \times 31 \times 12$	0.03125	0.5	Gauss
Wisconsin	GA	98.75	$6 \times 12 \times 5 \times 12$	0.1.875	0.375	Tanh
Adult	VNS	84.85	$6 \times 4 \times 6 \times 12$	0.140625	0.5	Logarithmic
Adult	SA	85.76	$6 \times 3 \times 19 \times 12$	0.09375	0.375	Sigmoid
Adult	GEO_{var}	83.26	$6 \times 3 \times 16 \times 12$	0.09375	0.25	Logarithmic
Adult	GA	84.51	$6 \times 3 \times 16 \times 12$	0.140625	0.5	Sigmoid
Glass	VNS	57.45	$6 \times 8 \times 4 \times 12$	0.75	0.5	Logarithmic
Glass	SA	61.12	$6 \times 2 \times 10 \times 12$	0.140625	0.375	Sigmoid
Glass	GEO_{var}	60.95	$6 \times 3 \times 16 \times 12$	0.46875	0.4375	Gauss
Glass	GA	63.86	$6 \times 27 \times 12$	0.125	0.5625	Logarithmic

In such experiments, the weighting factors given to training (ρ_1) and generalization (ρ_2) errors were equal to 1.0

Table 11 Test set accuracy rate (%) results including best ANN architectures encountered by VNS, SA, GEO, and GA algorithms

UCI data set	Metaheuristic	Test set accuracy rate (%)	Architecture	α	η	$\phi(\cdot)$
Wisconsin	VNS	98.68	$6 \times 3 \times 16 \times 12$	0.265625	0.1875	Tanh
Wisconsin	SA	97.75	$6 \times 14 \times 12$	0.453125	0.5	Sigmoid
Wisconsin	GEO_{var}	96.26	$6 \times 31 \times 11 \times 12$	0.25	0.5625	Tanh
Wisconsin	GA	99.25	$6 \times 4 \times 5 \times 12$	0.015625	0.6875	Sigmoid
Adult	VNS	85.91	$6 \times 4 \times 6 \times 12$	0.3125	0.1875	Tanh
Adult	SA	85.05	$6 \times 11 \times 14 \times 12$	0.28125	0.125	Sigmoid
Adult	GEO_{var}	83.60	$6 \times 5 \times 10 \times 12$	0.09375	0.25	Gauss
Adult	GA	84.96	$6 \times 7 \times 5 \times 12$	0.125	0.25	Logarithmic
Glass	VNS	57.92	$6 \times 30 \times 12$	0.09375	0.4375	Gauss
Glass	SA	59.61	$6 \times 16 \times 18 \times 12$	0.109375	0.626	Logarithmic
Glass	GEO_{var}	59.11	$6 \times 3 \times 15 \times 12$	0.21875	0.0	Tanh
Glass	GA	62.74	$6 \times 4 \times 5 \times 12$	0.09375	0.6875	Logarithmic

In such experiments, the weighting factors given to training (ρ_1) and generalization (ρ_2) errors were 0.1 and 1.0, respectively

Algorithm 1 Summary of the methodology employed for the feedforward ANN architecture optimization problem

```
   input  : A initial random solution $S$
   output: An optimized neural network architecture
 1 special treatment of the first solution;
 2 Build binary string randomly to represent the initial solution $S$;
 3 Initialize each ANN and perform the early stop training;
 4 Calculate the penalty, training error and generalization error values for each
   ANN;
 5 Calculate the objective function value for each trained ANN;
 6 $\mathbf{S^*} \leftarrow \mathbf{S}$;
 7 where $\mathbf{S^*}$ represents the best solution found until the moment;
 8 repeat
 9     Generate other candidate solution $\mathbf{S}$ based on the current solution by applying
       the metaheuristic;
10     Initialize each ANN and perform the early stop training;
11     Calculate the penalty, training error and generalization error values for
       each ANN;
12     Calculate the objective function value for each trained ANN;
13     if current solution $\mathbf{S}$ is better than $\mathbf{S^*}$ then
14         $\mathbf{S^*} \leftarrow \mathbf{S}$;
15 until the algorithm convergence or a maximum number of objective function
   evaluations be reached;
```

What we can infer about these and other results findings from the literature is the mere employment of any systematic and automated search method as proposed in this work is preferable to its absence.

Finally, we can say that although we applied the approach to solving specific problems like estimating air pollution sources, it is generic and robust enough to be adapted and applied to any problem that may be solved through ANNs.

7 Conclusion and final remarks

This paper proposes a new methodology for choosing feedforward ANN architecture with minimum complexity and optimal performance. We adapt and compare four global search metaheuristics to train and find out improved ANN architectures. The first case study consisted in to estimate correctly the intensity and location of the pollutants sources. Additionaly, more experiments were performed with three data sets from UCI repository. The network architecture selection considered not only the network training error but also the generalization error as a pruning criterion. Also, the generalization error is not simply measured but directly calculated during the evolutionary process. After that the architecture is trained by using a subset that does not belong to the test set. A new criterion to measure the ANNs model complexity based on the number of weights presents in the network arrangement and on the number of epochs needed for training is created and included in the methodology.

Based on the results obtained and those found in literature, we must highlight that it is fully justifiable to evolve the ANN architecture if our interest is to get a solution dedicated and adapted to the context of the problem. The time needed to evolve an architecture and find an appropriate setting for the problem is fully justified if compared with the time spent in an empirical "trial and error" procedure. Besides, we get near-optimal network architectures by using an autonomous, systematic, and well-behaved process, not only because of experience, intuition or even luck. The results also generate interesting conclusions on the importance of each input feature in classification and prediction tasks.

However, the computational cost translated into processing time for a monoprocessor machine to perform tens of thousands of objective function evaluations is still high and may become impractical if we adopt a greater granularity in the search space. A promising alternative would be to parallelize the optimization algorithm code and execute it in a multiprocessor environment, using the fact that ANNs are an intrinsically parallel algorithm. Another alternative would be the use of distributed parallel processing technology by grid computing where it would be possible to achieve a high processing rate by dividing the processing tasks among multiple machines. These processes would be executed when the machines were not being used by the user, avoiding the processing waste of the machine. In both cases, with minor code adjustments and improving the hardware, we could reduce the search time in more than one order of magnitude.

Acknowledgments The author is very grateful for the physical and financial support provided to this research by CAPES and CNPq.

References

1. Bernardos PG, Vosniakos GC (2004) Optimizing feedforward artificial neural network architecture. Eng Appl Artif Intell 20(3):365–382

2. Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. IEEE Trans Neural Netw 8(3):694–713
3. Rumelhart DE, Hinton GE, Willians RJ (1986) Learning internal representations of back-propagating error. Nat Biotechnol 323:533–536
4. Rojas R (1996) Neural networks, a systematic introduction, neural networks, a systematic introduction. Springer-Verlag, New York
5. Haykin S (1995) Neural networks. A comprehensive foundation, neural networks. A Comprehensive Foundation, Macmillan, New York
6. Goldberg DE, Holland JH (1988) Genetic algorithms and machine learning: introduction to the special issue on genetic algorithms. Mach Learn 3:95–99
7. Farmer JD, Toffoli T, Wolfram S (1983) Cellular automata. Proceedings of an interdisciplinary workshop at Los Alamos, New Mexico, March 7–11
8. Iyoda EM (2000) Inteligência computacional no projeto automático de redes neurais híbridas e redes neurofuzzy heterogêneas, Tese de Doutorado—Universidade Estadual de Campinas (UNICAMP)—Campinas, SP: [s.n.]
9. Miller GF, Todd PM, Hedge SU (1991) Designing neural networks. Neural Netw 4:53–60
10. Arifovic J (2001) Using genetic algorithms to select architecture of a feedforward artificial neural network. Physica A 289:574–594
11. Whitley D, Starkweather T, Bogart C (1989) Genetic algorithm and neural networks: optimizing connections and connectivity. Computing 14:347–361
12. Schafer JD, Caruana RA, Eshelman LJ (1990) Using genetic search to exploit the emergent behavior of neural networks. Physica D 42:244–248
13. Menczer F, Parisi D (1992) Evidence of hyperplanes in the genetic learning of neural networks. Biol Cybernet 66:283–289
14. Kitano H (1994) Evolution, complexity, entropy and artificial reality. Physica D 75:239–263
15. Kitano H (1990) Designing neural networks using genetic algorithms with graph generation system. Complex Syst 4:461–476
16. Harp S, Samad A, Guha A (1989) Toward the genetic synthesis of neural networks. In: Schafer JD (ed) Proceedings of the third international conference on genetic algorithms. Morgan Kaufman, San Mateo, CA, pp 762–767
17. Chen Z, Xiao J, Cheng J (1997) A program for automatic structure search. Proceedings of the 1997 international conference on neural networks, pp 308–311
18. Koehn P (1994) Combining genetic algorithms and neural networks: the enconding problem, master thesis, University of Tennessee, Knoxville, EUA
19. Bak P, Sneppen K (1993) Punctuated equilibrium and criticality in a simple model of evolution. Phys Rev Lett 71(24):4083–4086
20. Sousa FL, Vlassov V, Ramos FM (2003) Generalized extremal optimization for solving complex optimal design problems. Lect Notes Comput Sci 2723:375–376
21. Bak P (1996) How nature works. Copernicus, Springer-Verlag, New York
22. Hansen P, Mladenovic N (1995) A variable neighborhood algorithm: a new metaheuristic for combinatorial optimization, abstracts of papers at optimization days, 112
23. Hansen P, Mladenovic N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Dordrecht, pp 145–184
24. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Sci Agric 220(4598):671–680
25. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading
26. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Englewood Cliffs
27. Hornik K (1993) Some new results on neural network approximation. Neural Netw 6:1069–1072
28. Bishop C (1995) Neural networks for pattern recognition. Oxford University Press, Oxford
29. Ripley BD (1996) Pattern recognition and neural networks, pattern recognition and neural networks. Cambridge University Press, Cambridge
30. Roberti DR, Anfossi D, Campos Velho HF, Degrazia GA (2005) Estimating emission rate and pollutant source location. Cienc Nat (special volume):131–134
31. Ferrero E, Anfossi D (1998) Comparison of PDFs, closures schemes and turbulence parameterizations in lagrangian stochastic models. Int J Environ Pollut 9:384–410
32. Ferrero E, Anfossi D, Brusasca G, Tinarelli G (1995) Lagrangian particle model LAMBDA: evaluation against tracer data. Int J Environ Pollut 5:360–374
33. Luz EFP, Campos Velho HF, Becceneri JC, Roberti DR (2007) Estimating atmospheric area source strength through particle swarm optimization, inverse problems, desing and optimization symposium IPDO-2007, April 16–18, Miami (FL), USA, 1:354–359
34. Paes FF, Campos Velho HF, Carvalho AR, Luz EP (2008) Identifing atmospheric pollutant sources using artificial neural networks, AGU 2008 joint assembly, Fort Lauderdale (FL), USA, pp 131–134
35. Frank A, Asuncion A (2010) UCI machine learning repository [http://archive.ics.uci.edu/ml]. University of California, School of Information and Computer Science, Irvine, CA