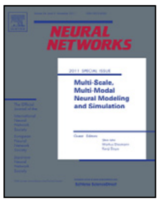
Neural Networks 26 (2012) 130–140

Contents lists available at SciVerse ScienceDirect

Neural Networks

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



C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons[☆]

José L. Subirats, Leonardo Franco[*], José M. Jerez

Departamento de Lenguajes y Ciencias de la Computación, E.T.S.I. Informática, Universidad de Málaga, Campus de Teatinos S/N, 29071, Málaga, Spain

ARTICLE INFO

Article history:
Received 19 April 2010
Received in revised form 15 April 2011
Accepted 10 October 2011

Keywords:
Constructive neural network
Incremental learning
Active learning
Generalization
Feed-forward network
Overfitting

ABSTRACT

C-Mantec is a novel neural network constructive algorithm that combines competition between neurons with a stable modified perceptron learning rule. The neuron learning is governed by the thermal perceptron rule that ensures stability of the acquired knowledge while the architecture grows and while the neurons compete for new incoming information. Competition makes it possible that even after new units have been added to the network, existing neurons still can learn if the incoming information is similar to their stored knowledge, and this constitutes a major difference with existing constructing algorithms. The new algorithm is tested on two different sets of benchmark problems: a Boolean function set used in logic circuit design and a well studied set of real world problems. Both sets were used to analyze the size of the constructed architectures and the generalization ability obtained and to compare the results with those from other standard and well known classification algorithms. The problem of overfitting is also analyzed, and a new built-in method to avoid its effects is devised and successfully applied within an active learning paradigm that filter noisy examples. The results show that the new algorithm generates very compact neural architectures with state-of-the-art generalization capabilities.

1. Introduction

Choosing the proper neural network architecture for a given classification problem remains a difficult issue (Baum & Haussler, 1989; Gómez, Franco, & Jerez, 2009; Lawrence, Giles, & Tsoi, 1996; Rumelhart, Hinton, & Williams, 1986), and despite the existence of several proposals to solve or alleviate this problem (Haykin, 1994), there is no general agreement on the strategy to follow in order to select an optimal neural network architecture. The computationally inefficient "trial and error" method is still much used in applications using Artificial Neural Networks (ANNs), but as an alternative different neural constructive algorithms have been proposed in recent years (Andree, Barkema, Lourens, Taal, & Vermeulen, 1993; Fahlman & Lebiere, 1990; Frean, 1990; García-Pedrajas & Ortiz-Boyer, 2007; Keibek, Barkema, Andree, Savenlie, & Taal, 1992; Mezard & Nadal, 1989; Nicoletti & Bertini, 2007; Parekh, Yang, & Honavar, 2000; Subirats, Jerez, & Franco, 2008; Utgoff & Stracuzzi, 2002). In general, constructive methods start with a small network (normally a single neuron in a single hidden layer) to then add new units as needed until a stopping criteria is met. The majority of the existing constructive algorithms freeze the value of older weights as learning proceeds, and thus what a neuron has learnt cannot be modified or improved in further stages. In this work, the Competitive MAjority Network Trained by Error Correction (C-Mantec) algorithm is introduced and applied to different sets of benchmark problems. The main novelty of the algorithm, in comparison to existing ones, is that C-Mantec incorporates competition between neurons and thus all neurons can learn at any stage of the procedure.

Competition among neurons has been much used in unsupervised learning for several applications like dimensional reduction, clustering, sparseness reduction, etc., both in artificial and biological plausible modeling (Hertz, Krogh, & Palmer, 1991; Intrator & Edelman, 1997; Piepenbrock & Obermayer, 1999; Rolls & Treves, 1998), but it has been less applied in supervised learning schemes (Grossberg, 1987). The new C-Mantec algorithm makes use of the thermal perceptron learning rule (Frean, 1992) that keeps the stability of the acquired knowledge permitting only modifications of the synaptic weights that will not lead the neuron to forget much of what it has learnt thus far, as it may occur with the standard perceptron rule when learning a nonlinearly separable function. In the C-Mantec algorithm neurons compete between them for learning the incoming information. The neuron that may need the smaller modification of its synaptic weight will learn according to the presented instance but only if the involved synaptic change is smaller than a previously set up parameter, i.e., one neuron is selected to

[☆] Preliminary results from parts of this work were presented at the 2008 ICANN conference: J.L. Subirats, L. Franco, I. Molina and J.M. Jerez, Active learning using a constructive neural network algorithm, Lecture Notes in Computer Science, 5164, pp. 803–811 (2008).

[*] Corresponding author. Tel.: +34 952 133304; fax: +34 95213 1397.
E-mail address: lfranco@lcc.uma.es (L. Franco).

0893-6080/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2011.10.003

learn the presented input but only if it is considered that the change of its synaptic weights will not disrupt what the neuron has learnt previously. If no such neuron is present in the actual architecture, a new unit is added and the network grows.

A severe problem affecting almost any predictive learning algorithms, and in particular neural network constructive ones, is the problem of overfitting (Caruana, Lawrence, & Giles, 2001; Geman, Bienenstock, & Doursat, 1992; Hawkins, 2004; Haykin, 1994). Models with fewer parameters are expected (following Occam's razor (Rodriguez-Fernandez, 1999)) to overfit less and thus there exist a kind of implicit competition in the field toward obtaining compact neural architectures. When the C-Mantec algorithm was applied to a set of noise-free Boolean functions it was observed that very compact architectures were generated and no overfitting effect was noticed. On the contrary, when the model was applied to real world data sets, in general containing noisy information, overfitting problems appeared. This is the standard situation for most of the existing constructive algorithms as many of them tries to achieve zero training error. To avoid overfitting problems when using C-Mantec on noise-contaminated data, a built-in method based on the analysis of the behavior of individual neurons was implemented and successfully tested on benchmark problems.

In this work, we introduce and analyze the C-Mantec algorithm and apply it to a wide set of classification problems, being able to show that the combination of competition, a stable learning procedure and a built-in method that prevents overfitting, can lead to the obtention of very compact neural architectures with good generalization abilities. Furthermore, a deep analysis of the behavior of the new algorithm was done as its parameters were modified, showing that C-Mantec is very robust on a wide range of their values and that they can very easily tuned.

2. The thermal perceptron learning rule

At the single neuron level the C-Mantec algorithm, to be described in detail later in Section 3, uses the thermal perceptron rule and thus we first give some details of this algorithm. The thermal perceptron, introduced by Frean (1992), is a modification of the original perceptron learning rule (Rosenhlatt, 1959) aimed to obtain a rule that provides a successful and stable linearly separable approximation to a non-linearly separable problem. The standard perceptron learning rule needs to be modified because it converges only when the problem is linearly separable and is unstable when applied to non-linearly separable problems.

Consider a neuron, modeled as a threshold gate, having two response states: ON = TRUE = 1 and OFF = FALSE = 0, receiving input from N incoming continuous signals. The activation state (S) of the perceptron depends on the N input signals, ψ_i, and on the actual value of the N synaptic weights (w_i) and the bias (b) as follows:

$$S = \begin{cases} 1\ (\text{ON}) & \text{if } \phi \geq 0 \\ 0\ (\text{OFF}) & \text{otherwise,} \end{cases} \tag{1}$$

where ϕ is the synaptic potential of the neuron defined as:

$$\phi = \sum_{i=1}^{N} w_i \psi_i - b. \tag{2}$$

In the thermal perceptron rule, the modification of the synaptic weights, Δw_i, is done on-line (after the presentation of a single input pattern) according to the following equation:

$$\Delta w_i = (t - S)\psi_i T_{fac}, \tag{3}$$

where t is the target value of the presented input, and ψ represents the value of input unit i connected to the output by weight w_i. The difference to the standard perceptron learning rule is that the thermal perceptron incorporates the factor T_{fac}. This factor, whose value is computed as shown in Eq. (4), depends on the value of the synaptic potential and on an artificially introduced temperature (T) that is decreased as the learning process advances, in a way that resembles the well known simulated annealing procedure (Kirkpatrick, Gelatt, & Vecchi, 1983).

$$T_{fac} = \frac{T}{T_0} \exp\left\{-\frac{|\phi|}{T}\right\}. \tag{4}$$

In Eq. (4), T_0 is the initial temperature value set at the beginning of the learning process, T is the actual temperature, and ϕ is the synaptic potential defined in Eq. (2). The value of the Temperature, T, is lowered steadily as the iterations (I) proceed, taking the value $T = 0$ when the maximum number of iterations ($I_{\max}$) is reached. We observed that it was not necessary to use different values of T_0 when the input dimension was fixed and thus all the experiments in this work have been run with a value of T_0 equals to the number of input variables, N.

The behavior of the thermal perceptron for large values of T (at the beginning of the learning process) is similar to the standard perceptron learning rule (PLR), as the value of T_{fac} is very close to 1. As the temperature gets reduced, the exponential factor in Eq. (4) increases its influence, making synaptic changes smaller. Synaptic weights are only modified when the perceptron output, S, is different to the target value, t, and thus for these wrongly classified inputs the absolute value of $|\phi|$ measures the distance to the correct classification region. For a given value of T, if $|\phi|$ is close to zero the exponential factor will be closer to 1, while it will be much smaller for large values of $|\phi|$. In this way, the exponential factor in Eq. (4) affects changes to the synaptic weights such as not to permit large changes if $|\phi|$ is large, and in this way prevents from forgetting the previous stored knowledge. The annealing schedule for the temperature permits larger synaptic modifications at the beginning of the process (exploration phase), while only minor adjustments can be done near the end, when the value of T is close to 0.

3. The C-Mantec algorithm

We introduce in this section the Competitive MAjority Network Trained by Error Correction (C-Mantec) algorithm. The algorithm constructs neural networks with a single hidden layer of neurons with threshold activation functions and a single output neuron. We consider first the case of single output functions, to later in Section 5 treat the multi-class case. The output neuron computes the majority function of the activation of the hidden units, as previous experiments have shown that the majority function has very good computational capabilities among the set of linearly separable functions (Subirats, Franco, Gòmez, & Jerez, 2008). The majority function, also called the median operator (Knuth, 2008), is a logic function from N input bits to one output. The value of the operation is false when less than half of the input bits are true, and true when half or more of the input bits are true. A further advantage of the majority function is that it can be easily implemented with a single output threshold neuron connected by unitary value weights to the hidden neurons and a threshold value equals to $N_h/2 - 1/2$ (N_h indicates the number of neurons present in the hidden layer).

The procedure for constructing an architecture for a given set of examples of dimension N starts by putting N inputs in the initial layer with the function to introduce the information into the network. Apart from these inputs, the initial architecture includes a single neuron in the hidden layer and an output neuron that will compute the majority function of the activation of the neurons present in the hidden layer. The learning process starts

```
1.  Create a one hidden layer neural network with a single neuron and a
output neuron computing the majority function.
2.  Set parameter values and initialize counters:
   $g_{fac} \in [0.05 - 0.5]$.
   $I_{max} \in [1000 - 100000]$ .
   $T_0 = N$ , $T = T_0$.
3.  Input a random pattern and check the output of the network.
4.  If the input example is not rightly classified then:
  4a.  Compute the value of $T_{fac}$ (Eq.  4) for all existing hidden neurons
that wrongly classify the presented input.
  4b.  Modify the weights of the neuron with the largest value of $T_{fac}$,
provided that this value is larger than the value of the parameter $g_{fac}$.
Lower the internal temperature of the modified neuron.
  4c.  If there is no neuron with a value of $T_{fac}$ larger than $g_{fac}$ then add a
new neuron to the hidden layer that learns the actual example, reset $T \to T_0$,
$I \to I_{max}$
5.  Loop back to instruction 3 until all patterns are classified correctly.
```

Fig. 1. Pseudocode of the C-Mantec algorithm.

as usual with the presentation of randomly chosen patterns from the training data set and the single present neuron in the hidden layer tries to learns these patterns using the thermal perceptron rule described in the previous section.

The C-Mantec algorithm has in principle 3 parameters to be set at the time of starting the learning procedure. Two of them are related to the thermal perceptron rule: the initial temperature value, T_0, and the maximum number of iterations allowed per learning cycle, $I_{\max}$, noting that each neuron in the hidden layer has its internal counter value I that will determine its temperature factor, T_{fac}. As we mentioned before, we have set the value of T_0 equals to N in all experiments and thus, for practical purposes only two parameters have to be adjusted. The third parameter is named the growing factor, (g_{fac}), as it determines when to stop a learning cycle and include a new neuron in the hidden layer.

Thus, the single neuron so far present in the architecture learns the input patterns according to Eq. (3) but only if the value of the temperature factor, T_{fac}, is larger than the set value of g_{fac}, one of the parameters of the algorithm. The first time this previous condition is not met, a new neuron is added to the hidden layer, the temperature of the two neurons set to T_0, and a new learning cycle starts with the iteration counter, I, set to 0. Thus, a learning cycle is the process occurring using a fixed number of neurons; when a new neuron is added a new cycle starts.

When more than one neuron is present in the architecture, the whole process is similar to the previous description, except for the fact that now competition arises between neurons: a random pattern is presented to the network, the activation state of the hidden neurons is computed, together with the whole network output state, S. If the output value, S differs from the target value of the input pattern, t, the algorithm then chooses among the neurons in the hidden layer that have wrongly classified this input pattern, selecting the one with the largest value of T_{fac}. If the value of T_{fac} for this selected neuron is larger than the set value of g_{fac}, this neuron modifies its synaptic weights according to the thermal perceptron equation (Eq. (3)) and its internal temperature is lowered by increasing its iteration counter. Note that each neuron has its own internal temperature value and thus a neuron T_{fac} depends on the value of the internal T and also on its actual value of ϕ. If there is no a neuron present in the architecture for which its T_{fac} is larger than the value of the g_{fac} parameter, the learning cycle is stopped, all neurons temperatures and iteration counters are reset and a new neuron is included in the hidden layer. Further, the new neuron tries to learn the lastly presented pattern helping to achieve convergence of the learning process. The maximum duration of a cycle is given by the value of $I_{\max}$ that determines the number of learning iterations for each of the neurons, but noting that in general the cycles are aborted and restarted whenever the condition $T_{fac} > g_{fac}$ is not met for the selected neuron.

In Fig. 1(a) pseudocode of the algorithm is shown, summarizing the most important steps of the C-Mantec algorithm and in Fig. 2(a) flow diagram of the algorithm is shown. In Fig. 2(a) filtering stage is included, showed at the lower right part of the figure (''Eliminate noisy examples''), that will be described later on in Section 4, and is indicated here to clarify at which point this filtering process is applied. The training process ends when all input examples are successfully classified, i.e., training is carried out until zero learning error is achieved. (Note that for the case in which a single hidden neuron is used, the output neuron can be eliminated and the final architecture is a simple perceptron).

The C-Mantec algorithm has in principle 3 parameters to be set at the time of starting the learning procedure. Two of them are related to the thermal perceptron rule: the initial temperature value, T_0, and the maximum number of iterations allowed per learning cycle, $I_{\max}$, noting that each neuron in the hidden layer has its internal counter value I that will determine its temperature factor, T_{fac}. As we mentioned before, we have set the value of T_0 equals to N in all experiments and thus, for practical purposes only two parameters have to be adjusted. The third parameter is named the growing factor, (g_{fac}), as it determines when to stop a learning cycle and include a new neuron in the hidden layer. From a conceptual point of view the effect of g_{fac} on the training process is not very different from that of T_{fac} in the annealing procedure, as both prevent learning wrongly classified patterns that are far from the actual separating hyperplane of a neuron, impeding the unlearning of the right classified patterns. We note that while T_{fac} acts as a modulation factor to the learning modification rule (cf. Eq. (3)), while g_{fac} acts as a fixed threshold, preventing any change on the synaptic values if there is no neuron for which T_{fac} is larger than g_{fac}. Having said that, from a practical and computational perspective, the effect of g_{fac} is quite important as directly controls the point at which new neurons are added to the hidden layer, affecting the final size of the architectures. This is, indeed, relevant because we have observed that in several cases minimum size architectures are not the best ones in term of the generalization ability obtained, and g_{fac} permits to control this directly. The other important fact is that as g_{fac} is independent of the temperature (unlike T_{fac}), its introduction makes the algorithm very robust in a wide range of values for the other two parameters of C-Mantec (the initial temperature and the number of iterations per cycle).

We could not demonstrate mathematically the convergence of the learning process for the C-Mantec algorithm as the use of a majority function as output, the implementation of competition between neurons, the possibility of modifying all synaptic weights during the whole training process, and the use of a stochastic

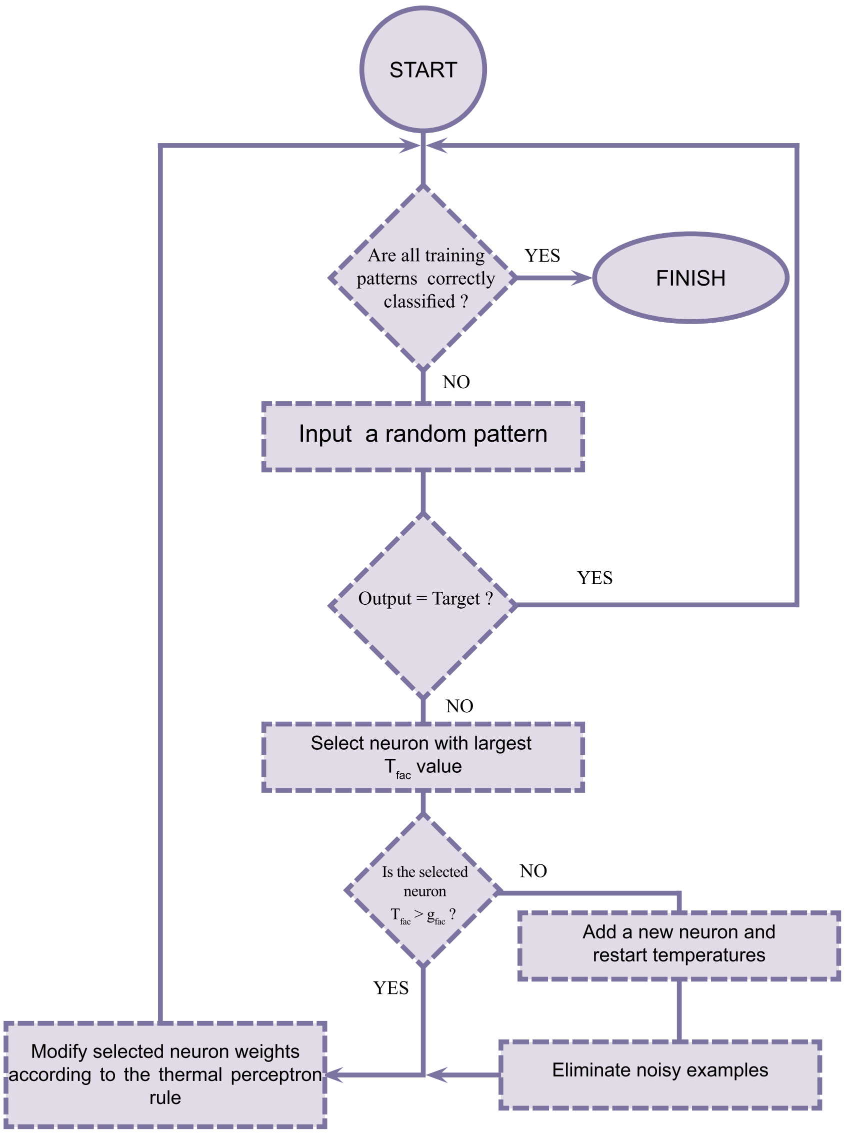

START
Are all training patterns correctly classified ?
YES
FINISH
NO
Input a random pattern
Output = Target ?
YES
NO
Select neuron with largest T_fac value
Is the selected neuron T_fac > g_fac ?
NO
Add a new neuron and restart temperatures
YES
Modify selected neuron weights according to the thermal perceptron rule
Eliminate noisy examples

Fig. 2. Flow diagram of the C-Mantec constructive learning algorithm where competition between neurons is introduced.

learning rule at the neuronal level complicate the application of the standard techniques normally used to prove convergence in constructive algorithms. Nevertheless, a strong convergence of the learning process has been observed in all analyzed cases and this can be explained by the fact that at a given time of the training process wrongly classified patterns can be learned by the addition of new units, while the use of a very stable learning procedure makes the neurons to learn only patterns close to their actual classification region preventing the unlearning of previous rightly classified patterns. Hypothetically, loops may arise during training as it seems possible that learning new patterns can cause unlearning previous ones and that the addition of new units may not be sufficient to prevent this. The recommendation in these eventual cases would be to check that the total error is not diminishing as new neurons are added (in order to confirm that a loop is occurring), and then modify the g_{fac} parameter toward larger values, as these changes will provide stability to the existing neurons while facilitating the inclusion of new neurons for learning the wrongly classified patterns. In combination to this, the total number of iterations can be reduced in order to speed up this phase. Once convergence is attained, we recommend to readjust the g_{fac} lowering its value to optimize the size of the generated architectures and the generalization ability, together with an increase on the number of iterations if necessary.

We refer also in this work to the Mantec algorithm, a preliminary version created during the development of the C-Mantec algorithm that corresponds to the algorithm operating with no competition between neurons, i.e., only weights related to the last introduced neuron can be modified during the learning procedure while the rest of the weights are frozen, in a similar way as most existing constructive neural networks algorithms works, and that serves to analyze clearly the role of competition, that we carry next.

3.1. *The effect of competition*

We run numerical simulations to test how the competition implemented in the C-Mantec algorithm affects its performance, measuring the size of the generated architectures and the generalization ability obtained. We test the C-Mantec algorithm against a version of the algorithm operating without competition, a version that will be refereed as the Mantec algorithm and also against the Upstart algorithm, introduced by Frean (1990), because it uses as well the thermal perceptron rule and does not implement competition. In a first experiment, we applied the three algorithms (C-Mantec, Mantec and Upstart) to the construction of neural architectures that implement a random

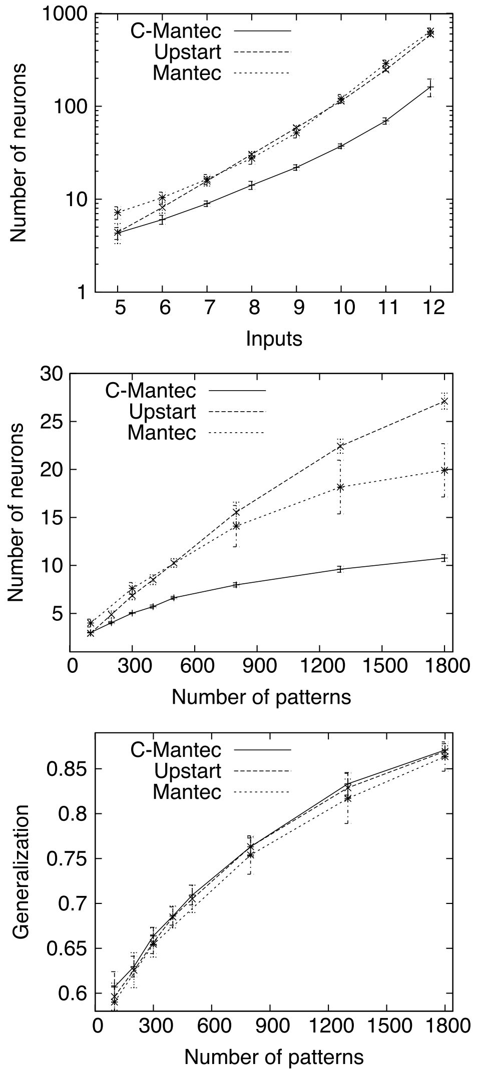


Fig. 3. Comparison between C-Mantec, Mantec and Upstart algorithms. Top graph: Number of neurons in the architectures generated by the three algorithms when learning random Boolean functions with a number of inputs between 5 and 12. Middle graph: Number of neurons of the generated architectures for the case of the two-or-more clumps function with $N = 25$ as a function of the number of patterns used for training. Bottom graph: Same as middle graph but results for the generalization ability obtained tested with 1800 additional patterns. (See the text for more details).

generated Boolean functions for different input size, N, between 5 and 12 using the whole set of available inputs (i.e., the network is trained with 2^N input–output pairings). The results are shown in Fig. 3 top graph, where it can be appreciated that the C-Mantec algorithm constructs more compact architectures than the other two methods as the number of inputs grows. For $N = 12$ the C-Mantec algorithm performs much better than the other two algorithms needing approximately only 160 neurons, almost one fourth of the approximately 600 neurons needed by the Upstart and Mantec algorithms (note that a logarithmic scale is used for the y-axis of the figure representing the number of neurons of the constructed architectures). We have not tested the generalization ability in this case as it has not much sense for totally random functions.

In a second experiment we have analyzed both the generalization ability and the size of the constructed architectures using the two-or-more clumps problem, used as a test function previously by several researchers (Denker et al., 1987; Frean, 1990; Mezard & Nadal, 1989; Utgoff & Stracuzzi, 2002; Wann, Hediger, & Greenbaum, 1990). This Boolean function outputs "1" for any input pattern having at least two groups ("clumps") of adjacent input bits ON. It is a middle-complexity function, easy to defined but relatively complex in the sense that N neurons are needed in the hidden layer to implement it for the case of N input bits. The training procedure was repeated 25 times using patterns generated through a Monte Carlo simulation in a similar way as used before in Denker et al. (1987) and Frean (1990). The obtained results (mean and standard deviation) are shown in Fig. 3 middle and bottom graphs for $N = 25$. Fig. 3 (middle graph) shows the performance of the three algorithms in terms of the size of the generated architectures, showing clearly that the C-Mantec outperforms both other algorithms. Regarding the generalization ability obtained, shown in Fig. 3 (bottom graph) the difference is not as large, as very similar performances are obtained for the three algorithms with a slight advantage in favor of C-Mantec, achieving approximately 0.5% of generalization above the results of Upstart, computed as an average across the different points shown as the number of patterns is increased between 100 and 1800. In the graph, the x-axis shows the number of training patterns used while the number of test patterns for computing the generalization ability was fixed and equal to 1800.

3.2. The role of the parameters g_{fac} and $I_{\max}$ on the size of the generated architectures and on the generalization ability obtained

In this subsection we analyze the behavior of the C-Mantec algorithm as the parameters g_{fac} and $I_{\max}$ are modified, with the aim of understand better the functioning of the algorithm and optimize its performance.

We run numerical simulations using as test function the two-or-more clumps function described in the previous subsection. Simulations were run with a number of inputs equals to $N = 8, 10, 12$, with g_{fac} taking the following values $\{0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55, 0.65, 0.7, 0.75\}$ and with a number of maximum iterations for learning cycle equals to the following values $I_{\max} = 1000 \times \{1, 3, 5, 10, 15, 20, 30, 50, 70, 100, 200\}$.

In Fig. 4 (top graph) three curves are plot showing the number of neurons in the hidden layer of the obtained architectures as a function of the g_{fac} parameter for the case $N = 12$. The three curves shown are for the mean across all the $I_{\max}$ values considered and for the minimum and maximum size networks found among all analyzed $I_{\max}$ values for a given g_{fac} value. As it can be seen the three curves are quite similar indicating the robustness of the algorithm regarding changes to $I_{\max}$ values. The behavior for the size of the architectures against changes in g_{fac} values is very clear, compact architectures are obtained for values lower than 0.65, and from this value on, the size of the architectures grows sharply. Fig. 4 (middle graph) analyzes the behavior of the generalization ability as the g_{fac} values are increased, and in this case a peak in the generalization ability for values between 0.55 and 0.7 is found. Values of g_{fac} larger than 0.7 produced a sharp decline of the generalization ability (not shown). In this figure, three curves are also shown for the mean, maximum and minimum value of generalization ability found across the range of $I_{\max}$ values considered, and again is possible to see a relative low spread between the curves. In order to have a more precise idea about optimal parameter setting, Fig. 4 bottom shows the optimal g_{fac} value found as the maximum number of iterations per cycle ($I_{\max}$) is modified, for the cases of achieving best generalization ability or minimum size networks. It seems clear from this graph, that different g_{fac} values might be used depending on the application required, i.e., a low g_{fac} is better for obtaining compact architectures while larger values of g_{fac} are preferred if the generalization ability is the relevant feature. It is an interesting result that smaller architectures does not lead to the best results in terms of the generalization ability, as it is suggested

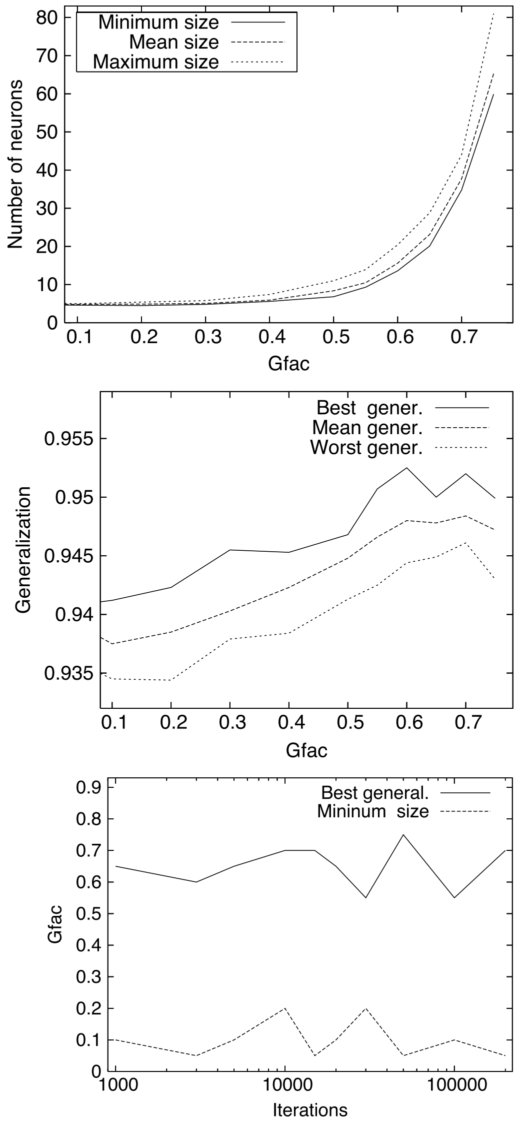


Fig. 4. Analysis of the role of the parameters $I_{\max}$ and g_{fac} on the generalization and the size of the architectures generated. The analysis is done using the function two-or-more clumps of 12 variables and averaging across 25 samples, using 75% of the available examples for training and 25% for measuring generalization. (a) The generalization ability obtained as a function of the temperature for the best g_{fac} value, for the mean of all values and for the worst values tested. (b) The number of neurons in the generated architectures for the three cases mentioned before. (c) The optimum g_{fac} values found for maximum generalization and minimum size as a function of the number of iterations.

by the Occam's razor principle (Rodriguez-Fernandez, 1999) and we will discuss this in the conclusions. The results for dimensions 8 and 10 were similar to those shown for the case $N = 12$.

Regarding the value of the parameters used for the rest of the experiments in this work, we note that we found that even if for low and middle complexity functions like the two-or-more clumps recently analyzed it is convenient to use values of $g_{fac} \sim 0.5$, a similar value was not a good choice for more complex functions as the algorithm generates larger networks, takes longer times and the generalization ability is not always optimized. For this reason smaller g_{fac} values between 0.05 and 0.1 were used (the exactly used values are indicated in the corresponding experiments).

4. Learning in presence of noisy examples: avoiding overfitting

During the several tests carried out with the C-Mantec algorithm, it was noted that when the algorithm was applied

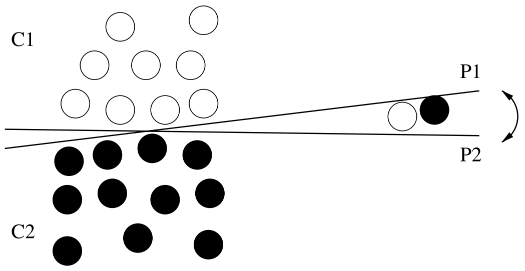


Fig. 5. Schematic drawing of the "resonance effect" that happens when noisy examples are present in the training set. The thermal perceptron will try several times to learn the contradictory examples (at the right of the figure) and this will produce an oscillation of the classification hyperplane. The number of times the synaptic weights are adjusted upon presentation of an example can be used to detect contradictory inputs.

to a set of logical "noise-free" functions (see Section 7.1) the generalization ability obtained was quite good in comparison to other algorithms, and this fact was taken as an indication that as very compact architectures are generated the algorithm does not suffer from overfitting (this type of overfitting is sometimes referred as "model overfitting") (Caruana et al., 2001; Geman et al., 1992; Hawkins, 2004; Haykin, 1994). Nevertheless, as the C-Mantec algorithm produces architectures that lead to zero training error, as most constructive algorithms do, it does overfit noisy examples when trained when real-world sets of problems. In order to avoid this "noise overfitting" effect, a procedure was introduced that consisted on eliminating input examples considered noisy. In general, a learning scheme in which there is some control over the inputs is known as active learning and it has been shown that can lead to an improved generalization ability (Angelova, Abu-Mostafa, & Perona, 2005; Bramer, 2002; Cohn, Atlas, & Ladner, 1994; Franco & Cannas, 2000).

The procedure to eliminate noisy inputs consists in analyze the number of times an example has been presented to the network and needed a synaptic weight correction (i.e., it was wrongly classified by the network), to eliminate at the end of a learning cycle those inputs that needed larger number of modifications in comparison to the mean. We have observed that a typical case in which the filtering scheme is needed is for the case of conflicting patterns, where two examples have similar input values but a different target value. In this case, a neuron trained by the thermal perceptron oscillates around possible solutions and we called this behavior the "resonant effect", that is illustrated schematically in Fig. 5. In the figure, the correct ("noise-free") set of examples is located in the left part of the figure, and on the right part the contradictory pair of inputs is shown. The "noise-free" set defines a beam of hyperplanes that can classify correctly its patterns but when the algorithm tries to learn the contradictory pair of inputs, it fails iteratively with one of the patterns, producing an oscillation of the proposed classification hyperplane. This oscillatory behavior normally finishes at the end of the learning cycle when the elimination process of "noisy" examples is carried out.

The implementation of the data pruning procedure is straightforward and consists in counting the number of times each individual example was presented and misclassified by the network, eliminating at the end of a cycle examples that have been presented the largest number of times. A parameter, n_{sig} (for "noise sigma"), is introduced and patterns that were presented to the network and wrongly classified a number of n_{sig} standard deviations larger than the mean are eliminated from the training set. The mean value is computed across the observed presentations of all training patterns. The parameter n_{sig} has to be adjusted and its optimal value will depend on the level of noise present in the data set, but several experiments with real world data indicated that a value between 2 and 3 was adequate to provide good results in terms of the generalization ability obtained. The experiments and results related to "noisy" data sets are included in Sections 7.2 and 7.3.

5. Extension of the C-Mantec algorithm to the treatment of multi-class problems

The C-Mantec algorithm was originally designed for working with binary classification problems and in order to extend its use to multiclass cases, we apply in this section three well known approaches normally used for tackling this problem. The three different extensions analyzed treat K-classes problems using a strategy that combines M binary neural networks using a simple additional decision module (Ou & Murphey, 2007). The M binary neural networks are trained separately using a common training data set or a subset of the original data, and a decision module is used to select the final classification results. The three different well known approaches used are One-Against-All (OAA), One-Against-One (OAO), and P-Against-Q (PAQ), that are briefly described below.

5.1. One-Against-All (OAA) neural networks

This scheme utilizes a $M = K$ binary neural networks, where K is the number of classes of the original task. Each neural network is trained with the same training data set but with different objective values. For an input x belonging to class i the target value of the neural network M_i is "1" for this input example and zero for all other $M - 1$ binary neural networks. In this scheme, there are three cases of possible outputs from the K binary networks to be analyzed by the decision module. First, if only one of the K neural networks showed "1" and all others output "0", the decision is easily made. Second, if more than one of the binary network output "1", the decision module needs to decide whether to output a symbol to indicate a "tie", or employs a more sophisticated scheme to force a classification decision. Third, if none of the neural networks outputs a "1", then the decision module needs to indicate that no classification is made. For the second case, in which more than one output module outputs a "1", our implementation prioritizes the neural network with the better classification rate obtained thus far, computed on-line. To resolve the third case, in which no output is produced by the binary modules, we select the class corresponding to the neural network with the worse classification rate computed up to the point when the decision is made.

5.2. One-Against-One (OAO) neural networks

This scheme transforms a K-Class pattern classification problem into $M = K(K - 1)/2$ binary sub-problems solved by binary neural networks. Each neural network solves a classification of an individual class against another individual class and is trained only with a subset of the data in which these two classes are present. The collective output from all M binary neural networks for an input feature vector 'x' represents a combination of M votes for K classes, and the decision module needs to decide the output class of the scheme. A simple voting scheme can be used by the decision module. This module counts the votes for each class based on the outputs from the M neural networks and the class with more votes is assigned to the input feature vector 'x'. The problem in this case is how the decision module resolves ties (equal number of votes for two or more classes). A tie can be broken by chosen the class with higher prior-probability, weighting the votes by the confidence value associated with each neural network output into account. In our implementation, ties where solved by restricting the data set to the classes involved and computing the voting only from these restricted cases. In some cases the tie cannot even be resolved and we just take in these cases a random choice between the involved classes. The major advantage of *OAO* approaches is that it provides some redundancy that can lead to a more robust system; the disadvantage is that generates a large number of neural network modules, specially when K is large.

5.3. P-Against-Q (PAQ) neural networks

The P-Against-Q classification scheme can be considered as a halfway approach between the other two mentioned previously. In this approach, the original classes are grouped in two classes a different number of M times, in a way that from the output of these groups is possible to infer the individual class. The implementation can be considered as M binary codes of length K, where each code has P bits equal to one and $Q = M - P$ bits equal to zero. One type of P-against-Q encoding consists in using the shorter code that specify all classes, $M = \log_2 K$ bits. This dense encoding is efficient in terms of the resulting size of the architecture, but it is not in terms of the generalization ability obtained, as some redundancy on the encoding is beneficial. In our implementation $2K$ modules were used and the grouping of the classes was randomly chosen.

6. Experiments

We analyze the performance of the C-Mantec algorithm through numerical simulations using two different sets of benchmark functions: a "noise-free" set of logical functions, frequently used in circuit design and a large set of problems belonging to the UCI repository coming from "real world" data. The analysis focuses mainly on the size of the constructed networks and on the generalization ability obtained.

7. Performance of the algorithm on obtaining compact architectures

We test in this section the capability of the C-Mantec algorithm to produce compact neural architectures by using a set of widely used circuit functions belonging to the MCNC benchmark. The set of 14 multi-output test functions used contains a number of input variables between 9 and 19 and were considered as 91 single output functions. The comparison of the size of the architectures produced by the C-Mantec algorithm is done against the results obtained using the DASG algorithm (Subirats, Jerez et al., 2008), as these constitute already a big improvement over previous published results, and as far as we know were the best results obtained at the time of its publication. The size of the architectures produced by the C-Mantec algorithm is shown in Table 1 together with the results produced by the DASG algorithm (Subirats, Jerez et al., 2008). Also shown in the last column of the table is the percentage of improvement of the C-Mantec algorithm over those obtained using DASG. In the last row of the table, the average results are shown. It can seen be that the C-Mantec algorithm outperform the results of the DASG algorithm by an average reduction of 39.08% in the total number of neurons included in the architectures. This significant reduction in the number of gates also implies a similar reduction in the fan-in max, maximum number of connections that a gate receives, that is a relevant factor at the time of constructing logic circuits.

7.1. Learning noise-free Boolean functions

A set of 17 single output Boolean functions from the MCNC benchmark was used to test the generalization ability of the C-Mantec algorithm operating in a noise-free environment. For this reason, the C-Mantec algorithm was executed without the active learning paradigm implemented for noisy data and described previously in Section 4. To compare the generalization performance of the C-Mantec algorithm, we analyze also other three well-known classification algorithms: the C4.5 decision tree algorithm (Quinlan, 1992), feed-forward neural networks (FFNNs) trained by backpropagation, and K-nearest neighbors algorithm for generalization (K-NN-gen). All these three alternative methods

Table 1
Number of neurons in the single layer of the constructed architectures obtained using the DASG and C-Mantec algorithms (columns 2 and 3 respectively) for a set of 14 multi-output Boolean functions. The first column indicates the name of the analyzed function and the number of inputs and outputs of the function is indicated in the second column. The last column shows in percentages the reduction in the number of neurons obtained by the C-Mantec algorithm in comparison to DASG.

Function name	I/O	Neurons DASG	Neurons C-MANTEC	Reduction (%)
9symml	9/1	21	3	85.71
alu2	10/6	96	51	46.88
x2	10/7	17	12	29.41
cm152a	11/1	9	8	11.11
cm85a	11/3	19	5	73.68
cm151a	12/2	18	6	66.67
alu4	14/8	279	150	46.24
cm162a	14/5	15	15	0.00
cu	14/11	22	18	18.18
cm163a	16/5	21	13	38.10
cmb	16/4	71	4	94.37
pm1	16/13	17	15	11.76
tcon	17/16	24	24	0.00
pcle	19/9	32	24	25.00
Average	–	47.21	24.86	39.08

Table 2
Generalization ability obtained with the new introduced C-Mantec algorithm and with other three standard algorithms, C4.5 decision trees, feedforward neural networks (FFNNs) trained with backpropagation and nearest neighbor for generalization K-NN-gen on a set of 17 Boolean functions. (See text for more details.)

Function	# Inputs	# Neur. C-Mantec	Generalization ability			
			C-Mantec	FFNN	C4.5	KNN-Gen
cm82af	5	3.00 ± 0.00	93.33 ± 11.11	90.00 ± 31.62	60.83 ± 38.10	65.00 ± 35.53
cm82ag	5	3.00 ± 0.00	60.00 ± 37.27	75.00 ± 28.60	37.50 ± 11.95	34.17 ± 23.06
cm82ah	5	1.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	57.50 ± 22.38	65.83 ± 23.06
z4ml24	7	3.00 ± 0.00	98.33 ± 3.67	100.00 ± 0.00	78.21 ± 6.84	81.28 ± 6.37
z4ml25	7	3.13 ± 0.74	90.83 ± 12.34	91.35 ± 12.00	56.47 ± 17.42	46.15 ± 13.45
z4ml26	7	3.00 ± 0.00	96.67 ± 5.89	92.76 ± 13.68	78.78 ± 18.12	80.19 ± 17.11
z4ml27	7	3.00 ± 0.00	99.17 ± 2.78	100.00 ± 0.00	83.72 ± 21.89	78.27 ± 26.75
9symml	9	3.00 ± 0.00	99.41 ± 0.86	96.67 ± 5.47	76.76 ± 4.56	75.02 ± 6.24
alu2k	10	11.21 ± 0.92	97.36 ± 1.90	84.39 ± 5.28	94.43 ± 4.21	90.25 ± 7.73
alu2l	10	18.9 ± 1.45	79.22 ± 5.54	75.79 ± 5.18	82.33 ± 3.55	78.03 ± 4.88
alu2o	10	11.16 ± 0.91	90.24 ± 2.27	90.04 ± 2.54	89.46 ± 3.58	88.67 ± 4.78
cm85al	11	1.00 ± 0.00	99.95 ± 0.16	100.00 ± 0.00	98.19 ± 0.66	98.73 ± 0.62
cm85am	11	3.00 ± 0.00	99.76 ± 0.42	99.41 ± 1.24	96.88 ± 0.25	96.78 ± 1.36
cm85an	11	1.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	98.34 ± 0.70	99.07 ± 0.81
alu4q	14	45.8 ± 3.62	99.72 ± 0.14	86.72 ± 2.06	98.80 ± 0.64	99.07 ± 0.21
alu4r	14	75.7 ± 0.98	97.73 ± 0.47	88.93 ± 1.39	96.45 ± 0.73	94.90 ± 1.18
alu4u	14	27.4 ± 0.99	99.39 ± 0.27	95.94 ± 0.37	98.16 ± 0.24	97.11 ± 0.35
Average	9.2	12.66 ± 20.2	94.18 ± 10.33	92.18 ± 8.17	81.34 ± 18.44	80.5 ± 18.97

were applied using the open source platform WEKA (Witten & Frank, 2000), using in most cases the standard parameter setting of WEKA, as this worked almost optimal in comparison to some alternative adjustment tested. A ten-fold cross-validation approach was used. The C-Mantec algorithm was run with parameter values: $g_{fac} = 0.05$ and $I_{\max} = 100{,}000$.

In Table 2 the results obtained from all 4 methods are shown, where it is possible to see that the best results where obtained by the new C-Mantec algorithm with an average generalization ability of 94.18%, followed closely by FFNN (92.18%). The application of the C4.5 decision tree algorithm leads to a generalization ability of 81.34% and similar results 80.50% of generalization were obtained with the K-NN-gen method. The difference between the generalization ability of the C-Mantec and the C4.5 and K-NN-gen methods was statistically significant at $p = 0.002$ and $p = 0.001$ respectively (paired t-test). For the comparison between C-Mantec and FFNN, the difference was not statistically significant ($p = 0.21$) as the difference in the average generalization ability obtained was much smaller.

7.2. *Learning real world problems*

We further test the performance of the C-Mantec algorithm on a set of 17 real world data sets obtained from the UCI machine learning repository (Blake & Merz, 1998). The set contains binary and multi-class problems with a number of inputs between 4 and 125 and a number of outputs between 2 and 19, and the results are shown in Table 3. The C-Mantec algorithm was used with values of $g_{fac} = 0.1$ and a number of iterations $I_{\max}$ equals to 100,000. The procedure for the elimination of noisy examples was applied in order to avoid overfitting problems using a value of 2.0 for the parameter n_{sig}. For the multi-output data sets analyzed, the C-Mantec algorithm was run with the P-against-Q (PAQ) approach, as this approach was shown to lead to the best generalization accuracy.

The first five columns in Table 3 show the name of the problem, the number of input and output variables, the number of neurons generated in the single hidden layer of the architectures and the generalization ability obtained with the C-Mantec algorithm. The last two columns show the generalization ability obtained from applying standard feed-forward neural networks (FFNNs) trained by backpropagation and support vector machines (SVMs). Both algorithms were run within the package WEKA using parameter settings equal or very close to the standard setting provided. The results shown in Table 3 come from a ten-fold cross validation approach used, and thus we report the mean and the standard deviation across the set of ten observed values. Average generalization ability values computed across the data set are shown in the last row of the table, where the standard

Table 3
Generalization ability obtained with the new C-Mantec algorithm and with other three standard algorithms, on a set of 17 real world problems from the UCI data set. (See text for more details.)

Function	Inputs	Classes	Neurons C-Mantec	Generalization C-Mantec	Generalization FFNN	Generalization SVM
Diab1	8	2	3.34 ± 1.11	76.62 ± 2.69	74.17 ± 0.56	76.80 ± 4.54
Cancer1	9	2	1 ± 0.0	96.86 ± 1.19	97.07 ± 0.18	96.68 ± 2.68
Ionosphere	34	2	2.00 ± 0.00	87.44 ± 0.06	91.06 ± 4.86	88.07 ± 5.32
Heart1	35	2	2.66 ± 0.74	82.63 ± 2.52	79.35 ± 0.31	83.86 ± 6.21
Heartc1	35	2	1.28 ± 0.57	82.48 ± 3.33	80.27 ± 0.56	84.80 ± 5.75
Kr-vs-Kp	40	2	3.00 ± 0.00	98.96 ± 0.62	99.34 ± 0.54	95.79 ± 1.34
Card1	51	2	1.78 ± 0.87	85.16 ± 2.48	86.63 ± 0.67	84.80 ± 5.75
Sonar	60	2	1.00 ± 0.00	75.00 ± 14.10	81.61 ± 8.66	73.61 ± 9.34
Mushroom	125	2	1.00 ± 0.00	99.98 ± 0.04	100.00 ± 0.0	100.00 ± 0.0
Iris	4	3	4.60 ± 0.80	96.00 ± 3.33	96.93 ± 4.07	96.27 ± 4.58
Bal-scale	4	3	12.40 ± 0.04	90.53 ± 3.74	90.69 ± 3.04	87.57 ± 2.49
Thyroid	21	3	5.00 ± 0.00	94.16 ± 0.51	96.85 ± 0.83	93.79 ± 0.31
Waveform	40	3	9.00 ± 0.00	86.50 ± 1.23	83.56 ± 1.66	86.68 ± 1.97
Horse1	58	3	9.40 ± 0.93	67.79 ± 5.71	66.79 ± 5.49	68.66 ± 5.16
Gene1	120	3	8.40 ± 2.70	86.24 ± 1.10	90.93 ± 1.80	90.96 ± 1.28
Glass	9	6	35.10 ± 3.50	67.00 ± 6.38	53.96 ± 2.21	57.36 ± 8.77
Soybean	82	19	24.00 ± 0.00	92.96 ± 2.68	90.53 ± 0.52	93.10 ± 2.76
Average			7.35 ± 9.25	86.25 ± 10.14	85.87 ± 12.41	85.81 ± 11.24

Table 4
Results for the number of neurons and generalization ability obtained with the C-Mantec algorithm using different multiclass schemes. (See text for more details.)

Function	Inp.	Cl.	N. OAA	N. OAO	G. OAA	G. OAO
Iris	4	3	6.60 ± 1.51	12.00 ± 0.00	95.33 ± 4.50	96.00 ± 7.16
Bal-scale	4	3	12.00 ± 0.00	12.40 ± 1.26	89.89 ± 4.67	91.72 ± 3.43
Thyroid	21	3	2.00 ± 0.00	3.00 ± 0.00	94.20 ± 0.60	94.20 ± 0.60
Waveform	40	3	14.90 ± 0.32	13.90 ± 1.28	85.22 ± 1.74	83.24 ± 1.35
Horse1	58	3	4.70 ± 0.70	3.00 ± 0.00	64.8 ± 6.40	66.20 ± 5.20
Gene1	120	3	5.20 ± 3.00	3.50 ± 1.10	84.00 ± 1.10	88.50 ± 1.00
Glass	9	6	11.40 ± 2.00	17.80 ± 1.70	58.40 ± 7.80	64.80 ± 6.30
Soybean	82	19	19.00 ± 0.00	171.00 ± 0.00	90.60 ± 3.20	92.60 ± 2.00
Average			9.48 ± 5.80	29.58 ± 57.42	82.81 ± 13.76	84.66 ± 12.45

deviation values indicated are computed from the values of generalization ability obtained for the individual problems. It is possible to see that the best results on average where obtained by the new C-Mantec algorithm with an average generalization ability of 86.24%, followed closely by FFNN (85.87%) and the SVM with a generalization ability of (85.81%), noting that the average difference between the three analyzed methods is not statistical significant. Surprisingly, the C-Mantec algorithm, that in average outperforms the other two methods, does not achieve better results in most individual cases but more constant ones. Regarding the computational times involved, we have measured the CPU times needed by the C-Mantec algorithm to compute two complex problems of the data set, the Waveform data set for which the C-Mantec needed 4693 s to complete the ten fold cross validation procedure applied, and the Kr-vs-Kp problem for which the algorithm needed 906 CPU seconds. As a comparison, a standard MLP needed 1766 and 632 CPU seconds respectively using the WEKA framework. CPU times were computed on a PC equipped with an Intel Core 2 2.13 GHz processor.

7.3. Multiclass approaches for the C-Mantec algorithm

We analyze in this subsection the performance of the C-Mantec algorithm in multi-class problems using three different multi-class schemes described before in Section 5. The data set for the tests consisted in 8 problems containing a number of classes between 3 and 19. The results obtained are shown in Table 4 for the OAA and OAO approaches, as the results for the PAQ scheme correspond to the ones already shown in the lasts rows of Table 3. For the set of multi-output problems analyzed, the generalization ability for the case of the PAQ approach was 85.11±11.48 and the number of neurons in the hidden layer of the architectures was 13.49±10.64 (mean values computed from the last 8 rows from Table 3). Thus, the PAQ approach leads to the best generalization ability (85.11%), followed closely by the OAO approach (84.66%), while the OAA approach leads to a much lower average value (82.81%). Regarding the size of the generated architectures, the smaller networks were obtained using the OAA approach as expected.

8. Discussion and conclusions

We have introduced in this work the C-Mantec constructive neural network algorithm for its application in supervised classification problems. The algorithm combines a local stable learning rule with global competition between all neurons in the hidden layer and thus it does not "freeze" weights connecting "old" introduced neurons, as it is the standard procedure in most existing constructive algorithms. An analysis of the properties of the algorithm has shown that it is very robust regarding changes in its parameter values which can be adjusted straightforward. We have further developed a built-in filtering paradigm that works by eliminating examples considered noisy and helped to overcome overfitting problems, permitting to obtain very good generalization values with "real world" data.

The capability of the C-Mantec algorithm for creating compact architectures was first thoroughly analyzed on the MCNC benchmark data set, widely used to test algorithms on logic circuit synthesis. The results presented in Table 1 clearly indicate an impressive synthesis capability of C-Mantec, outperforming by a 39.08% previously obtained results using the DASG algorithm (Subirats, Jerez et al., 2008). It is worth noting that the results obtained by the DASG algorithm were already an improvement on the results obtained by Zhang, Gupta, Zhong, and Jha (2005), and thus the present results constitute the best known ones obtained on these benchmark functions.

Regarding the prediction abilities of the new algorithm, we have tested them on two different sets of benchmark problems. First, we analyzed the prediction accuracy using 17 problems belonging to the MCNC benchmark set of logical functions, obtaining a better performance for C-Mantec in comparison to other three standard classification methods (FFNN, C4.5 decision trees and K-nearest neighbors). The average generalization ability obtained with C-Mantec was the largest one, even if it was very close to the value obtained with FFNN, but much larger that the obtained with the C4.5 and K-NN methods. A second test done to analyze the generalization ability of C-Mantec was carried out with "real world" benchmark problems containing binary and multi-class data. For this case, in order to avoid overfitting effects, the built-in active learning paradigm for selecting and eliminating examples considered as "noisy" was applied. The results confirmed that good levels of generalization can be achieved with results slightly above to those obtained with FFNN and SVM. Despite the fact that the values of generalization ability were quite similar among the three approaches, we note that the advantage of constructive algorithms against standard FFNN is that the choice of the architecture and the adjustment of the synaptic weights is automatically done during the training process. Regarding the three different approaches analyzed for the extension of the C-Mantec algorithm for the treatment of multi-class problems, the best results in terms of the generalization ability were obtained using the PAQ approach that also resulted in reasonable size architectures.

The obtained results show that the effect of implementing competition in the algorithm is beneficial and relevant for constructing more compact architectures but also for obtaining better generalization values. The rationale behind the fact of obtaining smaller architectures might be that as all neurons can learn at all times, they continue to incorporate knowledge during an extended period of time, fact that might be relevant also for dynamically changing data sets. In relationship to the generalization capability of the algorithm, the fact that all neurons can learn at all times makes the load of the neurons highly distributed, a clear difference with most existing constructive algorithms. In fact, Smieja (1993) highlighted the loading distribution between neurons as a factor that degrades the generalization ability of most constructive algorithms in comparison to the standard multilayer perceptron approach, that balances the learning among all neurons present in the hidden layers, as C-Mantec does.

As a final conclusion, we believe that the new C-Mantec algorithm not only is a useful and robust tool for constructing compact neural network architectures with good generalization abilities for classification problems, but also have been useful for showing how competition can be utilized in a simple way that might help in future practical and biologically motivated developments.

Acknowledgments

The authors acknowledge the support from MICIIN (Spain) through grants TIN2008-04985 and TIN2010-16556 (including FEDER funds) and from Junta de Andalucía through grant P08-TIC-04026.

References

Andree, H. M. A., Barkema, G. T., Lourens, W., Taal, A., & Vermeulen, J. C. (1993). A comparison study of binary feedforward neural networks and digital circuits. *Neural Networks*, *6*, 785–790.

Angelova, A., Abu-Mostafa, Y., & Perona, P. (2005). Pruning training sets for learning object categories. In *Proceedings of the 2005 IEEE computer society conference on computer vision and pattern recognition* (pp. 494–501).

Baum, E. B., & Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, *1*, 151–160.

Blake, C. L., & Merz, C. J. (1998). UCI Repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Bramer, M. (2002). Pre-pruning classification trees to reduce overfitting in noisy domains. *Lecture Notes in Computer Science*, *2412*, 7–12.

Caruana, R., Lawrence, S., & Giles, C. L. (2001). Overfitting in neural networks: backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*: *Vol. 13* (pp. 402–408). Denver, CO: MIT Press.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, *15*, 201–221.

Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., & Hopfield, J. (1987). Large automatic learning, rule extraction and generalization. *Complex Systems*, *1*, 877–922.

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*: *Vol. 2* (pp. 524–532). Los Altos, CA: Morgan-Kaufmann.

Franco, L., & Cannas, S. A. (2000). Generalization and selection of examples in feed-forward neural networks. *Neural Computation*, *12*, 2405–2426.

Frean, M. (1990). The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, *2*, 198–209.

Frean, M. (1992). Thermal perceptron learning rule. *Neural Computation*, *4*, 946–957.

García-Pedrajas, N., & Ortiz-Boyer, D. (2007). A cooperative constructive method for neural networks for pattern recognition. *Pattern Recognition*, *40*, 80–98.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*, 1–58.

Gómez, I., Franco, L., & Jerez, J. M. (2009). Neural network architecture selection: can function complexity help? *Neural Processing Letters*, *30*, 71–87.

Grossberg, S. (1987). Competitive learning: from interactive activation to adaptive resonance. *Cognitive Science*, *11*, 23–63.

Hawkins, D. M. (2004). The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, *44*, 1–12.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Macmillan: IEEE Press.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Addison-Wesley.

Intrator, N., & Edelman, S. (1997). Competitive learning in biological and artificial neural computation. *Trends in Cognitive Sciences*, *1*, 268–272.

Keibek, S. A. J., Barkema, G. T., Andree, H. M. A., Savenlie, M. H. F., & Taal, A. (1992). A fast partitioning algorithm and a comparison of binary feedforward neural networks. *Europhysics Letters*, *18*, 555–559.

Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Knuth, D. E. (2008). Introduction to combinatorial algorithms and Boolean functions. The Art of Computer Programming 4.0 (pp. 64–74). ISBN 0-321-53496-4.

Lawrence, S., Giles, C. L., & Tsoi, A. (1996). What size neural network gives optimal gener-alization? Convergence properties of backpropagation. *Technical report UMIACS-TR-96-22 and CS-TR-3617*. University of Maryland.

Mezard, M., & Nadal, J. P. (1989). Learning in feedforward layered networks: the tiling algorithm. *Journal of Physics A*, *22*, 2191–2204.

Nicoletti, M. C., & Bertini, J. R. (2007). An empirical evaluation of constructive neural network algorithms in classification tasks. *International Journal of Innovative Computing and Applications*, *1*, 2–13.

Ou, G., & Murphey, Y. L. (2007). Multi-class pattern classification using neural networks. *Pattern Recognition*, *40*, 4–18.

Parekh, R., Yang, J., & Honavar, V. (2000). Constructive neural-network learning algorithms for pattern classification. *IEEE Transactions on Neural Networks*, *11*, 436–451.

Piepenbrock, C., & Obermayer, K. (1999). The role of lateral cortical competition in ocular dominance development. In D. A. Cohn (Ed.), *Proceedings of the 1998 conference on advances in neural information processing systems II* (pp. 139–145). Cambridge, MA: MIT Press.

Quinlan, J. R. (1992). *C4.5: programs for machine learning*. CA: Morgan Kauffman.

Rodriguez-Fernandez, J. L. (1999). Ockham's razor. *Endeavour*, *23*, 121–125.

Rolls, E. T., & Treves, A. (1998). *Neural networks and brain function*. Oxford University Press.

Rosenhlatt, F. (1959). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–408.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by backpropagating errors. In D. Rumelhart, & J. Mc-Clelland (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition*: *Vol. 1* (pp. 318–362). MIT Press.

Smieja, F. J. (1993). Neural network constructive algorithms: trading general-ization for learning efficiency? *Circuits, Systems and Signal Processing*, *12*, 331–374.

Subirats, J.L., Franco, L., Gòmez, I., & Jerez, J.M. (2008). Computational capabilities of feedforward neural networks: the role of the output function. In *Proceedings of the XII CAEPIA'07, Vol. II* (pp. 231–238). ISBN:978-84-611-8848-2.

Subirats, J. L., Jerez, J. M., & Franco, L. (2008). A new decomposition algorithm for threshold synthesis and generalization of Boolean functions. *IEEE Transactions on Circuits and Systems I*, *55*, 3188–3196.

Utgoff, P. E., & Stracuzzi, D. J. (2002). Many-layered learning. *Neural Computation*, *14*, 2497–2539.

Wann, M., Hediger, T., & Greenbaum, N. N. (1990). The influence of training sets on generalization in feed-forward neural networks. In *Proceedings of the IJCNN international joint conference on neural networks. Vol. 3* (pp. 137–142).

Witten, I. H., & Frank, E. (2000). *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann Publishers, http://www.cs.waikato.ac.nz/~ml/weka.

Zhang, R., Gupta, P., Zhong, L., & Jha, N. K. (2005). Threshold network synthesis and optimization and its application to nanotechnologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *24*, 107–118.