Available online at www.sciencedirect.com
ScienceDirect
NEUROCOMPUTING
Neurocomputing 70 (2006) 351–361
www.elsevier.com/locate/neucom

Generalization ability of Boolean functions implemented in feedforward neural networks

Leonardo Franco*

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain

Received 15 July 2004; received in revised form 6 January 2006; accepted 10 January 2006
Communicated by M. Bianchini
Available online 21 June 2006

Abstract

We introduce a measure for the complexity of Boolean functions that is highly correlated with the generalization ability that could be obtained when the functions are implemented in feedforward neural networks. The measure, based on the calculation of the number of neighbour examples that differ in their output value, can be simply computed from the definition of the functions, independently of their implementation. Numerical simulations performed on different architectures show a good agreement between the estimated complexity and the generalization ability and training times obtained. The proposed measure could help as a useful tool for carrying a systematic study of the computational capabilities of network architectures by classifying in an easy and reliable way the Boolean functions. Also, based on the fact that the average generalization ability computed over the whole set of Boolean functions is 0.5, a very complex set of functions was found for which the generalization ability is lower than for random functions.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Perceptrons; Learning; Parity; Sensitivity; Complexity; Ferromagnetic systems

1. Introduction

Neural networks are nowadays widely used in different applications, mainly due to their property of generalization. Despite the existence of some general theoretical results [2,19,17,47,42–44,35] a complete understanding of the emergence of generalization remains unclear. In particular, the problem of selecting a network architecture for a determined application is still generally based on the "trial-and-error" method as no theoretical formula gives clear insight into this problem. In this paper, we introduce a simple measure for the complexity of Boolean functions that permits to classify the functions according to the generalization ability and training times that can be obtained when the functions are implemented in feedforward neural networks.

The complexity of Boolean functions has been studied for a long time [37], the aim of that being to have a measure for deciding if a problem is easier to solve or implement than another. In the 1960s a lot of work started in the field of threshold logic [46,20,10] where the synthesis of circuits using threshold units to compute Boolean functions were analyzed (see [28,45,29]). More recent works also analyzed methods for the decomposition of Boolean functions based on information relationship measures [7,34]. Issues related to the complexity of Boolean functions have been addressed within the area of circuit complexity [45,32], where the complexity of Boolean functions is analyzed in terms of their implementation on different kind of circuits, from those composed with logical AND/OR gates, to feedforward neural networks with threshold or sigmoidal activation functions. As efficient models for computation and also for their close relationship to neurophysiological models, a lot of work has been done on the study of the computational properties of neural networks [15,32,40,39,31], but despite the effort many aspects of their implementation and general properties remain unclear [2,17,24,35,12,6]. Statistical learning theory introduced by Vapnik and Chervonenkis [42] provided a mathematical framework for the study of the learning process and the generalization that can be expected in a

*Tel.: +34 952 13 3304; fax: +34 952 13 1397
E-mail address: lfranco@lcc.uma.es.

supervised training process (see [17,44,43]). The theory uses as a fundamental concept the VC dimension and many interesting and general results about generalization have been obtained. However, many practical issues remain unexplained and in particular the effect of the complexity of the target functions on the generalization ability has not been thoroughly explored.

Within the area of circuit complexity the complexity of a Boolean function has been defined as the minimum size of the circuit that could implement it, while the size of the circuit is measured as the number of nodes that composed the circuit. Also, the depth of the circuit, that is the number of layers or stages in which the computation is performed, is sometimes used as a measure of complexity, as it is more related to the overall computation time [45,32]. Many results have been derived for certain classes of functions, as for symmetric and arithmetic ones, where in general, bounds on the size of the circuits to compute the functions are obtained [40,32]. Moreover, other measures of complexity exists, depending on different interests, as for example, the total wire length has been recently introduced [25] as a more adequate measure to sensory processing systems and also related to the construction of VLSI circuits.

The previously mentioned measures have important theoretical and practical interest but, unfortunately, are very hard to calculate as almost all of them are related to the construction of optimal circuits.

There exists in computer science a measure, introduced in [23] and named average sensitivity, that is closely related to the measure proposed in this paper. Average sensitivity has been applied to calculate complexity bounds for Boolean circuits [5] and has also been related to the learning of certain classes of Boolean functions by using Fourier analysis [26] but up to our knowledge, no attempt was done on relating that measure to the generalization ability or training times obtained when the Boolean functions are implemented in feedforward networks. Some works where the complexity of the functions were discussed through their implementation in neural networks includes [24] where functions were generated with different complexities by selecting a parameter that controls the size of the maximum weight in the network, [30] where Boolean functions with different complexities were analyzed, and [33] where the learning process for noisy Boolean functions is analyzed.

Another interesting study was carried out in [11] where the complexity of Boolean functions was related to the difficulty observed in human concept learning. A recent survey of results concerning relationship between neural networks and Boolean functions can be seen in [1]. Boolean neural networks [3,21] are basically perceptron type networks that use only discrete weights and they have been also used to implement Boolean functions [9].

In next section the measure is formulated and analyzed; in Section 3 numerical studies are performed on neural networks with the aim of validating the proposal and finally the conclusions and possible extensions of this work are discussed.

2. A simple measure for the complexity of Boolean functions

A Boolean function, f, is a map $f:\{0,1\}^N \to \{0,1\}$, where N is the number of input bits. The function f is completely defined when the corresponding outputs for each of all the 2^N inputs are determined. We will refer in the rest of the paper by an *example* as one of the possible configurations of the input values together with its output, that is a string of $N+1$ bits. When a reference is given to neighbouring examples at Hamming distance 1 or 2, this means that the N bits defining the input of the example (not including the output) differs in 1 or 2 bits (i.e. for computing the Hamming distance only the N bits of the input to the network of each example are considered).

We propose as a valid measure for the complexity, $\mathscr{C}[f]$, of a Boolean function, f:

$$\mathscr{C}[f] = \mathscr{C}_1[f] + \mathscr{C}_2[f], \tag{1}$$

where $\mathscr{C}_1[f]$ and $\mathscr{C}_2[f]$ are the first and second order complexity terms taking into account pairs of examples at Hamming distance 1 and 2, respectively. For example, the first term, $\mathscr{C}_1[f]$, takes into consideration pairs of examples differing in just one input bit (nearest neighbour examples). The first term, $\mathscr{C}_1[f]$, can be explicitly written as

$$\begin{aligned}\mathscr{C}_1[f] &= \frac{1}{N_{\mathrm{ex}} * N_{\mathrm{neigh}}} \sum_{j=1}^{N_{\mathrm{ex}}} \\ &\quad \times \left(\sum_{\{l|\mathrm{Hamming}(e_j,e_l)=1\}} (|f(e_j) - f(e_l)|) \right) \\ &= \frac{1}{2^N * N} \sum_{j=1}^{2^N} \left(\sum_{\{l|\mathrm{Hamming}(e_j,e_l)=1\}} (|f(e_j) - f(e_l)|) \right),\end{aligned} \tag{2}$$

where the first factor, $1/(N_{\mathrm{ex}} * N_{\mathrm{neigh}})$, is a normalization one, counting for the total number of pairs considered, N_{ex} is the total number of examples equals to 2^N, and N_{neigh} stands for the number of neighbour examples at a Hamming distance of 1, as we are considering the first term. The indicated sums are performed first, over all the examples, and for every one of the examples, over all the N existing neighbouring examples at Hamming distance 1. The second term, $\mathscr{C}_2[f]$, has essentially the same structure as the first term, i.e., consists of a normalization factor, and a double summation, over all the examples and for each one of them over the neighbouring ones at a Hamming distance of 2.

$$\mathscr{C}_2[f] = \frac{1}{2^N * (N/2)} \sum_{j=1}^{2^N} \left(\sum_{\{l|\mathrm{Hamming}(e_j,e_l)=2\}} (|f(e_j) - f(e_l)|) \right). \tag{3}$$

Note that for the case $N = 2$ the definition of the complexity measure should include only the first term C_1. The reason for this is that for any example in dimension N (examples with N input bits) the number of neighbouring examples at Hamming distance i increases for $i = 1$ up to (Int)$N/2$ but then for Hamming distances from (Int)$(N/2) + 1$ up to N the number of neighbouring examples decreases. Given that the normalization used in the definition of the complexity measure is the total number of pairs of examples, the reduction in the number of neighbouring examples for Hamming distances larger than (Int)$(N/2) + 1$ implies that the effect of any pairs of examples with opposite inputs in the complexity measure increases as the Hamming distance between the pair increases and this is not right and thus the number of terms should be limited to (Int)$N/2$ (i.e. only the first term for the case $N = 2$) (see [14] for more details).

Note also that each term is normalized independently, through a normalization constant equals to the total number of pairs that exist at the Hamming distance considered. This normalization permits a maximum value per term of 1, that can be achieved by the first term but due to the structure of the examples space it is not possible for $\mathscr{C}_2[f]$ to reach this value. For example, there exists no function for which all the second nearest neighbours (Hamming distance = 2, between the inputs) have opposite outputs, as if for some examples their second nearest neighbours have opposite output, the same condition implies that there exist other examples for which this condition does not hold.

The definition of the complexity measure introduced in this work can be computed by counting the number of pairs of neighbouring examples having different outputs and weighting them according to the distance between the examples forming the pair, measuring how sensitive is the output of the function to changes in the input examples. The presented measure of complexity is inspired by the results obtained in [12,13], where a close relationship between the number of examples needed to obtain valid generalization and the number of neighbouring examples with different outputs was found. In those works a method for selecting examples in order to improve the generalization ability has been developed, and through its application to different functions and architectures scaling properties for the generalization ability of feedforward neural networks have been obtained.

The definition of the complexity measure contains two terms. Higher order ones can be incorporated but as a good correlation between the complexity measure (containing only the first two terms) and the generalization ability of the analyzed functions was found and also because simulations including higher order terms (third and fourth order terms were included) showed that the information provided by these terms is redundant (strong correlated values) with the information contained in the first and second terms, respectively.

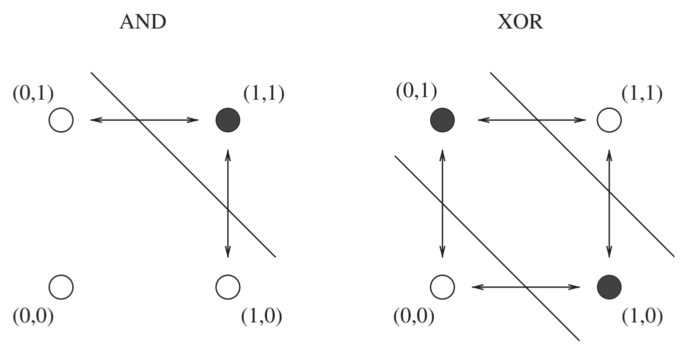


Fig. 1. Schematic drawing of the idea behind the complexity measure proposed in this work, exemplified on the AND and XOR functions for the case $N = 2$. The color of the circles, $(\text{white} = 0, \text{black} = 1)$, indicates the output value of the inputs, that are also indicated by a pair of numbers. The first term, C_1, of the complexity of the functions is proportional to the number of pairs of neighbouring examples having opposite outputs, and these pairs are indicated in the figure by segments ending with arrows.

The idea behind the proposed measure (for the case of the first term, C_1) is presented in a graphical form in Fig. 1 for the case of the first term, C_1, where the AND and XOR functions are shown for the case $N = 2$. The correspondent output values are indicated by empty (0) and filled (1) circles besides the inputs, that are also indicated by a pair of numbers. Regions with different outputs are separated by lines (hyperplanes in higher dimensions), as it is usually done to illustrate the idea of linearly separable functions [18,35]. In the figure, lines with arrows in their ends, and crossing the separating hyperplanes are drawn to indicate those neighbouring examples that have a different output. The value of the proposed measure of complexity is proportional to twice the number of these pairs of examples with different outputs, through a normalization factor equal to the total number of examples times the number of neighbours that each example has. In the presented cases of Fig. 1, the total number of examples is 4. Every example has two nearest neighbours at Hamming distance 1 and thus the value for the first term of the complexity measure, C_1, is equal to 0.5 for the AND function, and to 1 for the XOR function.

3. Validation of the proposal

In order to validate the usefulness of the proposed measure, different simulations were performed on feedforward neural networks to analyze how important features like generalization ability and training time are related to the measure. The simulations were done on one hidden layer architectures, fully connected between adjacent layers, applying back-propagation as learning algorithm. The choice is based on the fact that this combination of architecture plus algorithm is the most widely used and has been successfully applied to a large variety of problems [17,35].

In all the simulations the complexity measure was computed using the whole set of examples as the target function was known. In practice, one does not know the target function so the complexity measure should be computed from the training set. To analyze how the computation of the measure may be affected by using only the training set, we computed the values of the complexity measure for the case of all the Boolean functions with $N = 4$ inputs using half and two thirds of the total examples to obtain and average error of 5.1% and 4.3%, respectively, in comparison to the exact result, indicating that a good approximation is obtained. For cases with N larger than 4, we expect the same result, based on the idea that if the set of examples is representative of the function then this will apply as well to the complexity measure (For large N, if the selection of pair of examples used to compute the complexity measure is done randomly a 95% confidence interval value can be easily obtained, and the associated standard error will behave proportionally to $1/\sqrt{N_p}$ (N_p is the number of pairs of examples taken into account) [36].)

We focus our attention on the calculation of the generalization ability and on the time needed to train the system, without analyzing the computability of the implemented functions by the chosen architecture. The term computability, here and within this paper, refers to the task of finding a set of weights for the selected network architecture that correctly implements the mapping between a selected set of examples (inputs) and the targets (output) for a determined architecture, and it is also sometimes referred as the loading task or the learning problem (see [22,35]). This last mentioned task is a very complicated one, demonstrated to belong to the class of NP-complete problems [22,4,38], but very interesting issue, that deserves its own study. Even more, we think that the present work could serve as a useful tool to carry on a systematic study of the computational capabilities of neural networks.

For the implementation of the training process, the whole set of examples is split in three groups: training, validation and test sets. The size of these sets was kept constant in proportion to the number of total examples existing in each considered case: the training set contains half of the total number of examples selected at random, and the validation and generalization test sets have a fourth of the remaining examples each. It is considered, in a general sense, that the validation set forms part of the training set, as in some cases (not in this work) the network could be retrained including these examples. The training procedure consists of training the network using the training set by back-propagation, while monitoring the value of the error on the validation set, to find the point where its lower value is achieved. At that point the generalization error is measured using the test set of patterns. It was shown that through this procedure, the problem of overfitting can be reduced and a better generalization could be achieved [17,35]. Also, we calculate the computation time, measured in epochs, needed to reduce the training error to 0.25. In general, except where indicated, the obtained values are expressed as average results for group of functions with a complexity around the indicated value. Generalization ability and training error figures are given as a fraction of the correctly or wrongly classified examples, respectively. To be more precise, the generalization ability is always and only calculated on examples that have never been seen by the network.

We run simulations for all Boolean functions for the case of networks with $N = 4$ inputs, and for symmetric and randomly generated functions for the case with $N = 8$ inputs, using a variable number of neurons in the hidden layer. In the next subsections, characteristics and results of these simulations are described.

3.1. *Boolean functions with $N = 4$ inputs: an exhaustive study.*

In this subsection, we analyze all 65 536 Boolean functions that exist for the case $N = 4$, implementing them on a one hidden layer network containing seven neurons in the hidden layer. The question, whether or not the chosen structure could compute a determined function, is not analyzed here but from simulations performed on different network sizes, could be inferred that a high proportion of them can be computed by the used architecture. The lower values obtained for the training error at the end of the training process (750 epochs), shown in the last column of Tables 1 and 2, reinforces the previous hypothesis. We also calculate the generalization ability and the computational time needed to reduce the training error down to 0.25 (as the error is given in percentage correct, this means that 75% of the training examples were correctly classified). The results are presented in Tables 1 and 2, where the functions are grouped according to the complexity of the functions within the ranges indicated in the first column. In Table 1, the functions are grouped according to the first term of the complexity measure C_1, while in Table 2 the whole definition ($C = C_1 + C_2$) is used.

From the results, displayed in Table 1, it is possible to appreciate a high correlation between the values of the proposed measure of complexity and the practical features calculated to test the measure. The computed quantities behave as expected, given that for increasing complexity of the functions the generalization ability decrease monotonically, and the computational training time and final training error are larger for more complex functions. Only for the two more complex functions[1] in the range 0.9–1, the generalization ability value is higher than those obtained for less complex functions, this fact is not very significant as the value is calculated only on two functions. In Table 2, the results are shown for the case in which the functions are grouped according to the whole definition of the complexity measure, i.e. $\mathscr{C}[f] = \mathscr{C}_1[f] + \mathscr{C}_2[f]$. In this case, the

[1]These two functions are the parity and its negation function (NOT parity).

Table 1
Some properties obtained through a learning process for *all* the Boolean functions with $N = 4$ inputs, implemented in a feedforward 4-7-1 neural network architecture

Complexity range ($C = C_1$)	Number of functions	Generalization ability	Training time (epochs)	Final training error
0.0–0.1	2	1.00	1	0.000
0.1–0.2	96	0.88	13	0.005
0.2–0.3	424	0.83	42	0.005
0.3–0.4	8416	0.65	67	0.005
0.4–0.5	13 568	0.55	88	0.005
0.5–0.6	34 092	0.46	114	0.011
0.6–0.7	8416	0.41	137	0.017
0.7–0.8	424	0.40	158	0.022
0.8–0.9	96	0.35	195	0.029
0.9–1.0	2	0.50	208	0.062
Total or average	65 536	0.50	102	0.009

The complexity measure is calculated using only the first order term, i.e. $\mathscr{C}[f] = \mathscr{C}_1[f]$.

Table 2
Some properties obtained through a learning process for *all* the Boolean functions with $N = 4$ inputs, implemented in a feedforward 4-7-1 neural network architecture

Complexity range ($C = C_1 + C_2$)	Number of functions	Generalization ability	Training time (epochs)	Final training error
0.0–0.1	2	1.00	1	0.000
0.1–0.2	0	—	—	—
0.2–0.3	32	0.83	1	0.004
0.3–0.4	0	—	—	—
0.4–0.5	160	0.74	26	0.002
0.5–0.6	272	0.72	47	0.004
0.6–0.7	912	0.63	66	0.003
0.7–0.8	2312	0.62	71	0.003
0.8–0.9	7192	0.60	79	0.005
0.9–1.0	15 392	0.54	88	0.012
1.0–1.1	26 946	0.47	112	0.010
1.1–1.2	12 212	0.41	139	0.008
1.2–1.3	104	0.35	163	0.016
Total or average	65 536	0.50	102	0.009

The complexity measure is calculated using the two order terms, i.e. $\mathscr{C}[f] = \mathscr{C}_1[f] + \mathscr{C}_2[f]$.

generalization ability and training times show even better agreement with the complexity measure, as expected for a more accurate measure, evidenced by less dispersion around the mean values, that we measured for both cases through the calculation of the standard deviation. The obtained values for the standard deviation, averaged across all functions, were 0.238 and 0.230 for $\mathscr{C} = \mathscr{C}_1$ and $\mathscr{C} = \mathscr{C}_1 + \mathscr{C}_2$, respectively. Furthermore, it is shown in Fig. 2 the Pearson correlation coefficient, R, computed between the generalization ability obtained for each of the 65 536 Boolean functions with $N = 4$ inputs and the value of complexity measure computed for each function, written as a function of α, $\mathscr{C}[f] = \mathscr{C}_1[f] + \alpha\, \mathscr{C}_2[f]$. $\alpha = 1$ corresponds to the case $\mathscr{C} = \mathscr{C}_1 + \mathscr{C}_2$, that is the definition of complexity used in this work and for which the highest correlation value of $R = -0.668$ (statistically significant at $p < 0.0001$) was obtained.

It is worth noting that the total average generalization ability obtained is 0.50, a property that also holds in general cases whenever the average is done on the whole set of Boolean functions. In Section 3.4 we demonstrate this property and discuss some implications arising from it.

3.2. Symmetric functions

An important class of Boolean functions are the symmetric ones, those with values independent of the order of the input, i.e. the output depends only on the total number of input bits ON (the number of 1's). The class of symmetric functions contains many fundamental functions like sorting and all types of counting ones, including also the well-known parity function [45,32,35]. They have been extensively studied and many lower bounds have been obtained for circuits computing them. A general result

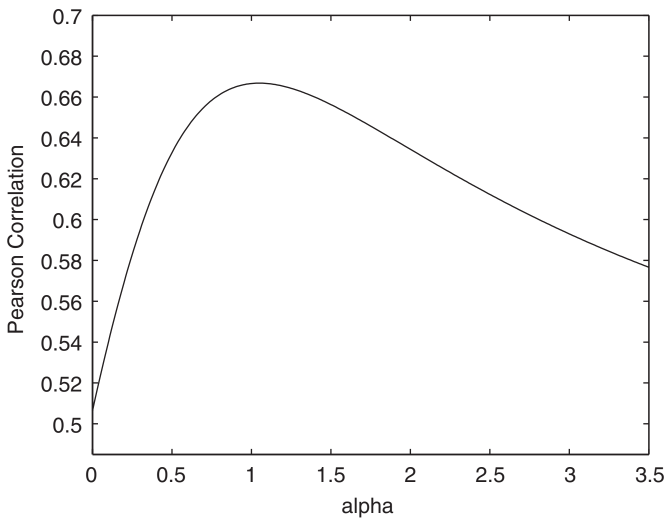


Fig. 2. Absolute value of the Pearson correlation coefficient computed between the generalization ability obtained for each of the 65 536 Boolean functions with $N = 4$ inputs and the value of the complexity, computed as a function of α, $\mathscr{C}[f] = \mathscr{C}_1[f] + \alpha\mathscr{C}_2[f]$. $\alpha = 1$ corresponds to the case of the definition of complexity used in this work and for which the highest correlation value of 0.668 ($p < 0.0001$) was obtained.

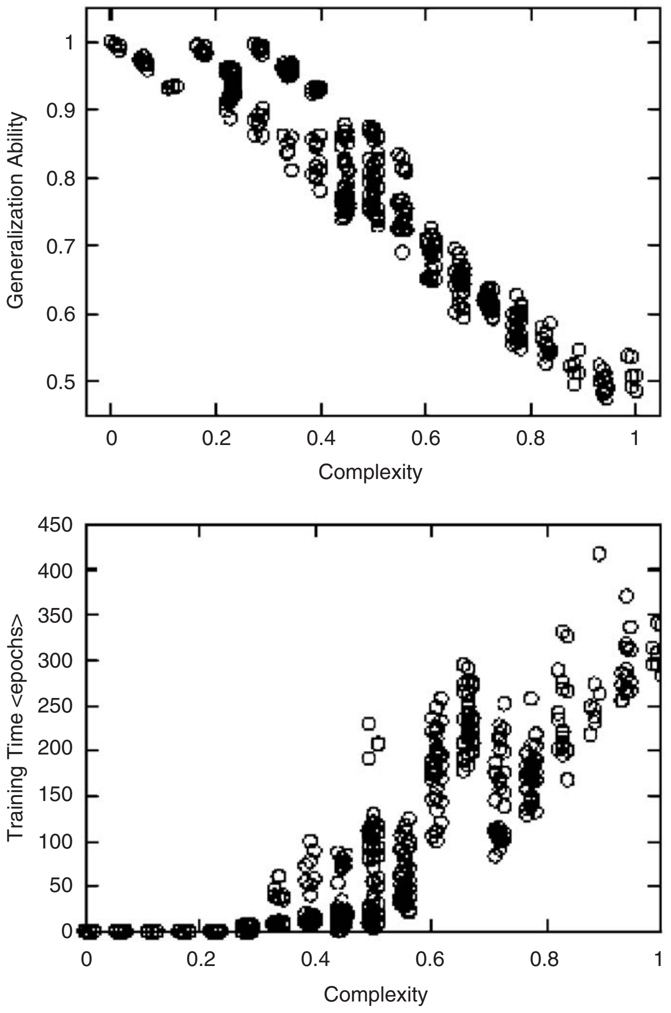


Fig. 3. Generalization ability (a), and training time (b) vs. complexity (first order) for all the symmetric functions with $N = 8$, in a 8-29-1 neural network architecture. The generalization error is measured through a validation procedure, in which the net is trained on half the total number of examples, 128, while monitoring the value of the error on other 64 examples, and at the best validation point the generalization error is measured on the remaining 64 examples. The training time is measured at the point when the training error is reduced to 0.25. Learning was implemented through standard backpropagation.

states that a circuit of size $\mathcal{O}(\sqrt{N})$ gates and depth 3 with polynomial weights is enough to compute all the symmetric functions [40]. The number of symmetric functions for the case of N input bits is 2^{N+1}, since the set of input examples can be divided in $N + 1$ groups, with 0 to N activated inputs.

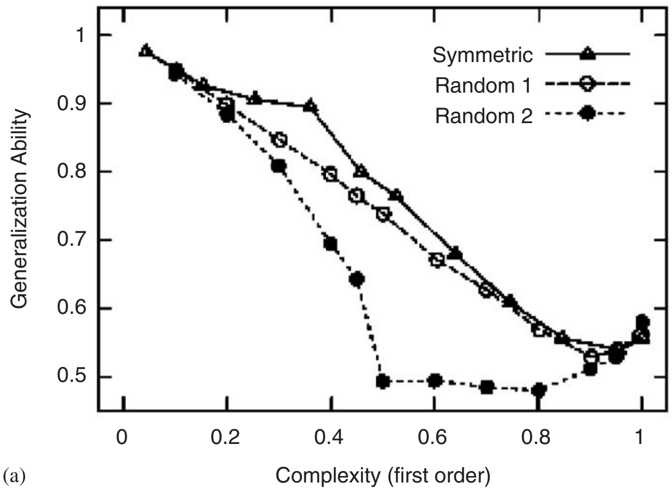
We carry on simulations with all the 512 symmetric functions for the case $N = 8$, using neural networks with one hidden layer with a variable number of hidden neurons within the range from 2 to 29. The results for the generalization error and computational time are plotted vs. the complexity (only the first order used) in Figs. 3a and b. Certain variability exists for functions with the same complexity, but in an average sense, a good agreement is found, as the level of complexity increases, the generalization ability diminishes, while the training times increase.

It is also shown in Figs. 4a and 5 the behaviour of the generalization ability vs. the complexity for symmetric functions when using the whole definition of the complexity measure (first and second order included), in comparison to the results obtained for random generated functions (these results are discussed in next subsection when the random generated functions are analyzed). Also in Fig. 4b the values for the training time vs. the complexity (first order) are plotted for the mentioned classes of functions (note that the learning algorithm used was standard backpropagation as different training times might be obtained with different training procedures).

3.3. *Random functions*

There exists 2^{2^N} Boolean functions of N inputs, making their exhaustive study very complicated except for very simple and small cases, like the case $N = 4$ studied in a previous subsection. The problem of generating sample functions to represent the total distribution of functions is not very simple, as for example, trying to generate random functions by a random assignment of outputs ends with a set of functions, all with a complexity (order one) around 0.5, as the probability of obtaining a function with a different complexity is very low.

For the case $N = 8$, we generate two sets of random functions: Random 1 and Random 2 sets. The set Random 1 is generated in two parts: the simpler functions are generated by modification of the constant function, that has all output values equal to 0. Thus, we take the initial configuration and modify the output values randomly with a certain probability, to obtain functions with a complexity



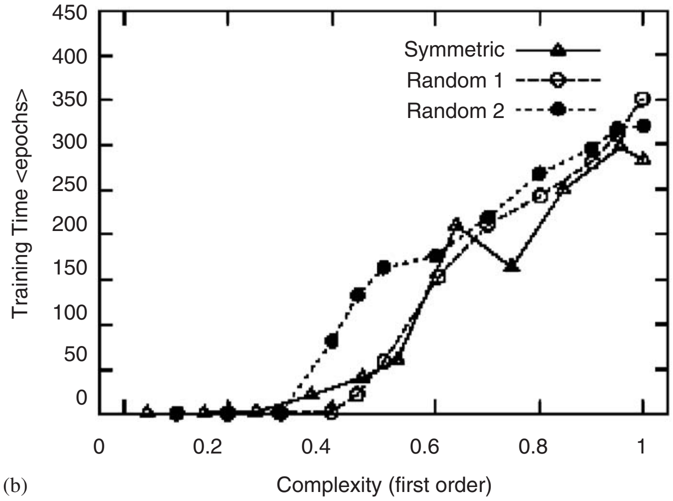


Fig. 4. Generalization ability (a) and training time (in epochs) (b) vs. complexity (first order) for three different generated families of Boolean functions with $N = 8$ inputs (see text for details).

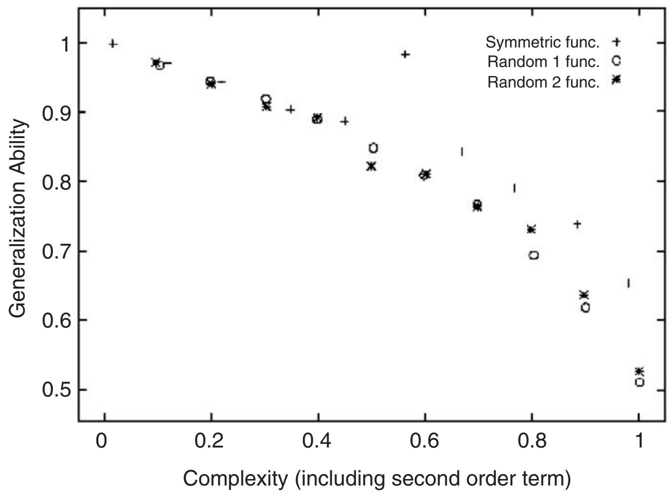


Fig. 5. Generalization ability vs. complexity (first + second order terms considered) for three different generated families of Boolean functions: Random1, Random 2, and Symmetric functions (see text for details).

(order one considered) from zero to 0.5. The other part, covering the range of complexities from 0.5 to 1 is generated in the same way but taking as initial configuration the parity function. The parity function [41,13] is a very complex and well studied function, belonging to the class of symmetric functions. The flipping probabilities used to create the Random 1 set of functions were: (0.047, 0.099, 0.149, 0.280, 0.500, 0.750, 0.833, 0.857, 0.950, 0.997) for the 10 groups with complexity ($C = C_1$) between 0.1 and 1.0.

The Random 2 set, is generated straightforwardly in a single way process, starting from the parity function as initial configuration, and by flipping with certain probability only the bits that are equal to 1 in the definition of the parity function. In this way, we obtain functions similar to the parity function when the probability of flipping bits is very low, but when this probability of flipping increase to higher values functions similar to the constant function are obtained, and thus, functions that cover the whole range of complexities (order one) from 1 to 0 are generated. The flipping probabilities used in this case were: (0.091, 0.167, 0.286, 0.333, 0.375, 0.474, 0.500, 0.545, 0.583, 0.928). In Figs. 4a and b we show the results obtained for the two sets of random generated functions and also for comparison, the results corresponding to symmetric functions are shown. Within each class of functions a good agreement between the complexity measure and the calculated feature (generalization ability and computation time) is obtained, but between different classes we observe that for functions with the same complexity a different generalization ability on average is obtained.

A better match between the generalization ability for the different classes of functions is obtained if the whole definition of the complexity measure (including the first and second order terms of the expansion (see Eq. (1)), i.e. $\mathscr{C} = \mathscr{C}_1 + \mathscr{C}_2$) is used. The calculation now involves up to pairs of second order nearest neighbour and some extra computational cost, as there exist $N * (N - 1)/2$ second nearest neighbours for each example. In Fig. 5 the results are shown for the two sets of random generated functions Random 1 and 2 and also for the symmetric functions. As it could be seen from the figure, a good agreement is obtained for the generalization ability values for functions with the same complexity for both sets of Random functions. The Pearson correlation coefficient, R (and the significance level, p, of the correlation), was computed for the three set of results showed in Fig. 5 obtaining for the case of the symmetric functions $R = -0.892$ ($p < 0.001$). For the two sets of Random generated functions, R_1 and R_2 the values for the correlation coefficient were $R = -0.970$ ($p < 0.0001$) and $R = -0.969$ ($p < 0.0001$), respectively.

3.4. *A general result about average generalization and the existence of very complex functions*

From a comparison of the results obtained for all the functions with $N = 4$ and those for random generated functions with $N = 8$, a clear difference in the values of the average generalization error could be noticed. While for the $N = 4$ case, this value was found to be 0.5 (see Table 1), for the case $N = 8$ a value greater than 0.5 would be obtained, as a consequence that for all the considered functions the generalization error is greater than 0.5 (see Figs. 4a and 5). We do not calculate this value because the calculation would involve using the probability distribution of functions (according to their complexity) which we do not know.

To solve this apparent controversy, we realize that the fact of having obtained a value 0.5 for the generalization ability of all Boolean functions for the case $N = 4$ is a general property arising whenever the average of the generalization ability is computed on the whole set of Boolean functions, a result that we state as the following lemma:

Lemma. *For any fixed architecture device, that could be trained by examples, as for example a neural network, the average generalization error on a complete set of Boolean functions is 0.5 independent of the learning algorithm used.*

Proof. Consider first, the case in which the network is trained on $2^N - 1$ examples, and the generalization ability is tested on the remaining example. All the possible training sets (2^N different sets) are considered assuming that the network is trained only once on each different training set (this is not a restriction for the demonstration but makes it easier).

Thus, the average generalization error could be written as

$$\mathscr{E}_{\mathrm{g}} = \frac{1}{2^{2^N} 2^N} \sum_{\text{all functions}} \times \sum_{\text{all training sets}} |Target_i - Output_i|, \qquad (4)$$

that we rearrange as a sum over pairs of functions sharing $2^N - 1$ training examples (input and output), but differing in the remaining one. Now the generalization error can be written as

$$\mathscr{E}_{\mathrm{g}} = \frac{1}{2^{2^N} 2^N} \sum_{\text{coupled pairs}} |Output_i - 1| + |Output_{\bar{i}} - 0|, \qquad (5)$$

where i indicates an example and $\bar{i}$ denotes another example sharing the same training set. The value of $|Output_i - 1| + |Output_{\bar{i}} - 0|$ is 1 (for all the terms) because the training set is the same in both cases and thus one of the two terms is always right (value of 0) while the other is wrong (value 1). Eq. (5) contains half the number of terms of Eq. (4), as we grouped the terms in pairs, having exactly $(2^{2^N} 2^N)/2$ terms equal to 1, half of the normalization factor, and thus the value 0.5 for the average generalization error is obtained.

The generalization of the proof for the general case in which the number of training examples is $2^N - i$, $i = 1, \ldots, 2^N - 1$, can be done in a similar way by construction. As said, the training set now contains $2^N - i$ examples and thus there exist 2^{2^i} Boolean functions compatibles with this training set, as we can assign to the remaining i examples (not belonging to the training set) all possible combinations of outputs. This set of Boolean functions, is such that for every example not belonging to the training set half of the times the output of the functions is 1 and half of the times is 0. Thus, when the generalization error is computed and averaged across all these compatible Boolean functions, half of the functions will agree with the target value of each of the test examples and half will disagree, giving a generalization error of 0.5. It is worth noting that this result does not depend on the level of error achieved during the training or in the size of the training set, as it arises from properties of the structure of the set of existing Boolean functions compatible with the training set. This set can be said to be "symmetric", in the sense that exists equal number of positive (output 1) and negative (output 0) examples, and this is why when the average generalization error is computed the result is 0.5. $\square$

At the light of this result, it seems clear that the set of functions generated randomly and analyzed for the case of $N = 8$ input bits, was not totally representative of the whole set of functions. Moreover, the lemma implies that this set of ignored functions should be very complex and that the value expected for the average generalization ability should be lower than 0.5.

From an analysis of the way in which the functions used in the case $N = 8$ were generated, it is possible to see which are the functions that have not been considered. This "new" set could be generated, again by starting from the parity function, but by flipping not individual output bits, but strings of contiguous output bits when the examples are ordered in a certain way, for example according to the number of inputs bits ON, i.e. regions or domains are created where the function seems to be the parity function, but in another region the outputs are equal to those of the NOT parity function. Through the mentioned process we obtain functions with a measure of complexity (including the second order term) greater than 1, in general within the range from 1 to 1.3 (for the case $N = 8$). When these functions are implemented in a neural network the generalization ability that can be obtained is lower than 0.5, implying that these functions are harder to predict than completely random functions. The generalization ability of these very complex functions was also analyzed by implementing them in different size networks. Different architectures with a number of neurons in the hidden layer from 1 to 45 were constructed, finding in all cases an average generalization ability below 0.5 that confirms the high complexity of these functions. Similar results for the case of time series implemented in simple perceptrons were obtained by Zhu et al. [48]. They show that given a predictable sequence it is always possible to find another "antipredictable" sequence, for which the generalization ability is 0. In the case considered in this paper with functions and not sequences, the "antipredictable" functions appear as a consequence of the lemma demonstrated above, and does not imply that if for a function a generalization ability of 1 is found then there exists another complementary function with a generalization ability of 0: an average generalization ability of 0.5 is obtained when the whole set of functions is considered and not when pairs of functions are considered.

4. Conclusions

From ideas taken from the calculation of the minimum number of examples needed for perfect generalization, applied to different functions and architectures [12,13], a simple measure permitting the classification of Boolean functions according to their complexity was proposed (Eqs. (1)–(3)). The measure is found to be highly correlated with the generalization ability obtained from the implementation of the functions in neural networks architectures. The main advantage of the measure in comparison to other existing ones, like those within the area of circuit complexity [32], is the fact that the introduced measure could be easily computed from the definition of the function itself, independently of its implementation.

The measure was tested on the whole set of Boolean functions for $N = 4$, and on different classes of Boolean functions, like symmetric and randomly generated ones, for the case $N = 8$, for different network sizes. In all cases a strong correlation was obtained between the proposed measure and practical features at the time of the implementation of the functions on neural networks, as generalization ability and training times. On average, the generalization ability decreases monotonically with the complexity of the functions, while the training times increase, as it is expected for more complex functions. Moreover, for the case when all the symmetric functions with $N = 8$ inputs were considered, the generalization ability and training time show a nice agreement with the proposed measure, for almost *all* functions, as for functions with the same complexity similar values of the generalization ability and training times were obtained (ref. Fig. 3). The correlation between function complexity and the generalization obtained was quantified by measuring the Pearson correlation coefficient, R. For the case of all Boolean functions with $N = 4$ inputs the value obtained was $R = -0.668$, while for the symmetric functions the value was $R = -0.892$. In the cases of the random generated functions for $N = 8$ the correlation found was even higher, $R = -0.970$ and $R = -0.969$ for the two sets considered, confirming what can be observed from the results presented in Tables 1 and 2 and in Fig. 5.

The proposed measure consists of two terms accounting for the effect of pairs of neighbour examples with different outputs, weighted by the Hamming distance between the examples. For functions belonging to the same class (according to the way in which the functions are generated, see Section 3 for details), the first order complexity measure, $\mathscr{C}_1$, was enough to establish a nice correspondence between the generalization ability and the complexity value proposed, but when the comparison was done for functions with the same complexity value but belonging to different classes (see Fig. 4), some discrepancies appeared, making it necessary to use the second order term of the series, $\mathscr{C}_2$, in order to obtain similar generalization values between different groups of functions (see Fig. 5).

When the whole set of Boolean functions was analyzed for the case of $N = 4$ input bits, an average value of 0.5 was obtained for the generalization ability. We proved that this property is a general one for any value of N, valid whenever the average of the generalization ability is calculated on the whole set of Boolean functions. Furthermore, this last mentioned result led us to discover the existence of a set of highly complex functions, for which a generalization ability below 0.5 was predicted and confirmed by testing them on different architectures.

It is worth mentioning that an analogy exists between the complexity measure and the Hamiltonian of a system of spins with ferromagnetic interactions [14] fact that might be used to obtain a better understanding of the complexity structure of the Boolean functions, as results and tools developed within statistical mechanics can be used.

A question, related to the proposed measure, that arises is whether the presented complexity measure has any relationship with other complexity measures related to Kolmogorov complexity [8,27]: in a general sense, the proposed complexity measure has a relationship, as the most complex functions are the less predictable ones, those for which a low generalization ability is obtained. Considering a function as a string of the 2^N output bits, as it would be the case of applying measures related to the Kolmogorov one, would be less accurate than the proposed measure, as the measure introduced in this work takes into account the structure of the space of examples, by establishing a measure of the influence of pairs of examples on the generalization ability, related to the Hamming distance between the examples, and thus having, in a general sense, more information about the complexity of the functions. In a sense, the complexity measure proposed in this paper is in agreement with a desirable property of complexity measures [16] that "complexity" should be used for something that is between plain uniformity and total randomness (while Kolmogorov complexity is more related to randomness).

In light of the results, we conjecture that the interesting relationship found between the complexity measure and the generalization ability could be exploited as a valuable tool for exploring the complex computational capabilities of neural networks. We also note that the results in the present work were obtained by numerical simulations in relatively small architectures and for some classes of Boolean functions. We think that the results will extrapolate to larger architectures, but more rigorous work might be needed in order to fully understand the present results and to generalize them to higher dimensions.

Acknowledgements

L.F. thanks Roberto Zecchina and Martin Anthony for fruitful discussions and Sergio A. Cannas and Steven Young for a critical reading of the manuscript. L.F. acknowledges support from the Spanish Ministry of Education and Science through a Ramón y Cajal fellowship, and from CICYT (Spain) through Grant TIN2005-02984 (including FEDER funds).

References

[1] M. Anthony, Boolean functions and artificial neural networks. CDAM Research Report LSE-CDAM-2003-01, in: Y. Crama, P. Hammer (Eds.), Boolean Functions: Volume II, 2006, to appear.

[2] B.E. Baum, What size of neural net gives valid generalization?, Neural Comput. 1 (1989) 151–160.

[3] S. Bhide, N. John, M.R. Kabuka, A Boolean neural network approach for the travelling salesman problem, IEEE Trans. Comput. 42 (1993) 1271–1278.

[4] A.L. Blum, R.L. Rivest, Training a 3-node neural network is NP-complete, Neural Networks 5 (1992) 117–127.

[5] R.B. Boppana, The average sensitivity of bounded-depth circuits, Inform. Process. Lett. 63 (1997) 257–261.

[6] R. Caruana, S. Lawrence, C. Lee Giles, Overfitting in neural networks: backpropagation, conjugate gradient, and early stopping, Advances in Neural Information Processing Systems, MIT Press, Denver, 13 (2001) 402–408.

[7] A. Chojnacki, L. Jozwiak, An effective and efficient method for functional decomposition of Boolean functions based on information relationships measures, in: Proceedings of Design and Diagnostics of Electronic Circuits and Systems, DDECS'2000, Smolenice, Slovakia.

[8] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley, New York, 1991.

[9] V. Deolalikar, Mapping Boolean functions with neural networks having binary weights and zero thresholds, IEEE Trans. Neural Networks 12 (2001) 639–642.

[10] M.L. Dertouzos, Theshold Logic: A Synthesis Approach, MIT Press, Cambridge, 1965.

[11] J. Feldman, Minimization of Boolean complexity in human concept learning, Nature 407 (2000) 572–573.

[12] L. Franco, S.A. Cannas, Generalization and selection of examples in feedforward neural networks, Neural Comput. 12 (2000) 2405–2426.

[13] L. Franco, S.A. Cannas, Generalization properties of modular networks implementing the parity function, IEEE Trans. Neural Networks 12 (2001) 1306–1313.

[14] L. Franco, S.A. Cannas, Non glassy ground-state in a long-range antiferromagnetic frustrated model in the hypercubic cell, Physica A 332 (2004) 337–348.

[15] L. Franco, J.M. Jerez, J.M. Bravo-Montoya, Role of function complexity and network size in the generalization ability of feedforward networks, in: Proceedings of IWANN'05, Vilanova i la Geltrú, Spain, Lectures Notes in Computer Science, vol. 3512, 2005, pp. 1–8.

[16] P. Grassberger, Information and Complexity Measures in Dynamical Systems, in: Information Dynamics, Plenum press, New York, 1991, pp. 15–33.

[17] S. Haykin, Neural Networks: A Comprehensive Foundation, Macmillan, New York, 1994.

[18] J. Hertz, A. Krogh, R.G. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley, Redwood City, CA, 1991.

[19] S.B. Holden, P.J.W. Rayner, Generalization and PAC learning: some new results for the class of generalized single layer networks, IEEE Trans. Neural Networks 6 (1993) 368–380.

[20] J. Hopcroft, R. Mattson, Synthesis of minimal threshold logic networks, IEEE Trans. Electron. Comput. EC-14 (1965) 552–560.

[21] C. Ji, D. Psaltis, Capacity of two-layer feedforward neural networks with binary weights, IEEE Trans. Inform. Theory 44 (1998) 256–268.

[22] J.S. Judd, Neural Network Design and the Complexity of Learning, MIT Press, Cambridge, MA, 1990.

[23] J. Kahn, G. Kalai, N. Linial, The influence of variables on Boolean functions, in: Proceedings 29th IEEE Symposium on the Foundations of Computer Science (FOCS' 88), 1988, pp. 68–80.

[24] S. Lawrence, C.L. Giles, A.C. Tsoi, What size neural network gives optimal generalization? Convergence properties of backpropagation, Technical Report UMIACSTR -96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, 1996.

[25] R.A. Legenstein, W. Maass, Foundations for a circuit complexity theory of sensory processing, in: Advances in Neural Information Processing Systems, MIT Press, Denver, vol. 13, 2001, pp. 94–110.

[26] N. Linial, Y. Mansour, N. Nisan, Constant depth circuits, Fourier transform and learnability, J. ACM 40 (1993) 607–620.

[27] L. Lovász, Information and complexity (how to measure them?), in: B. Pullman (Ed.), The Emergence of Complexity in Mathematics, Physics, Chemistry and Biology, Pontifical Academy of Sciences, Princeton University Press, Vatican City, 1996, pp. 65–80.

[28] S. Muroga, Threshold Logic and its Applications, Wiley, New York, 1971.

[29] A.L. Oliveira, A. Sangiovanni-Vincentelli, LSAT: an algorithm for the synthesis of two level threshold gate networks, in: Proceedings of the ACM/IEEE International Conference on Computer Aided Design, Santa Clara, CA, IEEE Computer Society Press, Silver Spring, MD, 1991, pp. 130–133.

[30] A.L. Oliveira, A. Sangiovanni-Vincentelli, Learning complex Boolean functions: algorithms and applications, in: Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann, Denver, 1993, pp. 911–918.

[31] P. Orponen, Computational complexity of neural networks: a survey, Nordic J. Comput. 1 (1994) 94–110.

[32] I. Parberry, Circuit Complexity and Neural Networks, MIT Press, Cambridge, MA, 1994.

[33] M. Perkowski, L. Jozwiak, S. Mohamed, New approach to learning noisy Boolean functions, in: Proceedings ICCIMA'98—International Conference on Computational Intelligence and Multimedia Applications, Gippsland, Victoria, Australia, World Scientific, Singapore, 1998, pp. 693–706.

[34] M. Rawski, L. Jozwiak, T. Luba, Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures, J. Syst. Architect. 47 (2001) 137–155.

[35] R.D. Reed, R.J.II. Marks, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, MIT Press, Cambridge, MA, 1999.

[36] J.A. Rice, Mathematical Statistics and Data Analysis. 2nd ed., Duxbury Press, Belmont, CA, 1994.

[37] C.E. Shannon, The synthesis of two-terminal switching circuits, Bell Syst. Tech. J. 28 (1949) 59–98.

[38] J. Sima, Back-propagation is not efficient, Neural Networks 9 (1996) 1017–1023.

[39] J. Sima, P. Orponen, General-purpose computation with neural networks: a survey of complexity theoretic results, Neural Comput. 15 (2003) 2727–2778.

[40] K.Y. Siu, V.P. Roychowdhury, T. Kailath, Depth-size tradeoffs for neural computation, IEEE Trans. Comput. 40 (1991) 1402–1412.

[41] C. Thornton, Parity: the problem that won't go away, in: Proceedings of AI-96, Toronto, 1996, pp. 362–374.

[42] V.N. Vapnik, A.J. Chervonenkis, The necessary and sufficient conditions for consistency of the method of empirical risk minimization, Pattern Recognition Image Anal. 1 (1991) 284–305.

[43] V.N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.

[44] M. Vidyasagar, A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems (Communications and Control Engineering), Springer, Berlin, 1997.

[45] I. Wegener, The Complexity of Boolean Functions, Wiley, New York, 1987.

[46] R.O. Winder, Threshold Logic, Ph.D. Dissertation, Department of Mathematics, Princeton University, 1962.

[47] D.H. Wolpert (Ed.), The mathematics of generalization, in: Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning, Santa Fe Institute Studies in the Sciences of Complexity, Volume XX, Addison-Wesley, Reading, MA, 1995.

[48] H. Zhu, W. Kinzel, Antipredictable sequences: harder to predict than random sequences, Neural Comput. 10 (1998) 2219–2230.

Leonardo Franco was born in Tucumán, Argentina. He did masters and Ph.D. studies in physics at the University of Córdoba, Argentina analyzing the generalization properties of feedforward neural networks. He moved later as a postdoctoral fellow to SISSA, Trieste, Italy where he became involved in Computational Neuroscience. He then moved to the University of Oxford, UK, as a research scientist where he applied and developed information theory methods for the analysis of neuronal recordings from the visual system. He is at present at Málaga University, Spain as a Ramón y Cajal researcher working on neural networks, their applications to biomedical problems and also in computational neuroscience. He has authored about 25 publications in journals and international conferences. He is a member of the IEEE.