# Generalization Properties of Modular Networks: Implementing the Parity Function

Leonardo Franco and Sergio Alejandro Cannas

***Abstract*—The parity function is one of the most used Boolean function for testing learning algorithms because both of its simple definition and its great complexity. Being one of the hardest problems, many different architectures have been constructed to compute parity, essentially by adding neurons in the hidden layer in order to reduce the number of local minima where gradient-descent learning algorithms could get stuck. We construct a family of modular architectures that implement the parity function in which, every member of the family can be characterized by the fan-in max of the network, i.e., the maximum number of connections that a neuron can receive. We analyze the generalization ability of the modular networks first by computing analytically the minimum number of examples needed for perfect generalization and second by numerical simulations. Both results show that the generalization ability of these networks is systematically improved by the degree of modularity of the network. We also analyze the influence of the selection of examples in the emergence of generalization ability, by comparing the learning curves obtained through a random selection of examples to those obtained through examples selected accordingly to a general algorithm we recently proposed.**

***Index Terms*—Circuit complexity, generalization, learning from examples, modular neural networks, parity function.**

## I. Introduction

One central theme in neural networks is the design of optimal architectures to solve specific problems. The implementation of an optimal feedforward network for a given problem involves several aspects, such as complexity of the problem, depth of the network (i.e., number of hidden layers), number of neurons in the hidden layers, convergence of the learning algorithms, ability to generalize, etc.

In this work we consider networks that solve the parity function using feedforward neural networks composed of linear threshold units: neurons whose activity $\sigma_i$ is determined by computing a linear threshold function of the form

$$\sigma_i = \Theta\left[\sum_j J_{ij} S_j - T_i\right] \tag{1}$$

where

| | |
|---|---|
| $\Theta(x)$ | 1 if $x \geq 0$ and 0 otherwise; |
| $J_{ij}$ | synaptic weights; |
| $S_j = \{0, 1\}$ | activities of neurons in a previous layer; |
| $T_i$ | activation threshold of the neuron $\sigma_i$. |

Different complexity measures as order of predicate, entropy decreasing criteria, size of the network, etc., (see [2], [5], and [13]) together with the fact that every change of a single input bit produces a change in the output make the parity function a hard problem among Boolean functions. The parity function is thus one of the most used functions for testing learning algorithms, because of its simple definition but great complexity.

From the vast literature where the parity function is analyzed and compared to another functions we could cite [5], [9], [10], [12], [16], and [18].

The question about what is the minimal size network needed to compute parity has been addressed from the point of view of circuit complexity, see [13]. Impagliazzo *et al.* [11] have found that the $N$ bit parity function with a single hidden layer needs at least $N^{1/2}$ hidden neurons, while the best known construction has $\mathcal{O}(N)$ neurons. Recently in [8], it was demonstrated up to $N = 4$, that the minimum size of the hidden layer required to solve the $N$-bit parity is $N$.

This network, that we shall refer to as the basic structure, is the simplest and most studied architecture to compute the parity function (see [10], and [12], [16]). It has $N$ input bits, $N$ fully connected neurons in a single hidden layer, and a single output that has to be ON whenever an odd number of input bit are ON and OFF otherwise. Gradient-descent learning algorithms face problems with this architecture as they get trapped in local minima, so other solutions have been designed by adding neurons in the hidden layer to obtain an improvement in the performance of the learning procedure [17].

In this work we construct a family of modular architectures that implement the parity function. Every member of the family can be characterized by the fan-in max ($f_{\max}$) of the network, i.e., the maximum number of connections that a neuron can receive. This parameter controls the degree of modularity, the number of hidden layers, which increases logarithmically with $f_{\max}$ and the total number of synaptic weights $N_s$. An interesting fact is that $N_s$ diminishes as $f_{\max}$ increases.

Besides several computational advantages (see for example [3], [9] and references therein) modular architectures are of particular interest from the biological point of view. Modularity seems to be an important principle in the architecture of vertebrate nervous systems, while fully connected networks are rarely found in nature. Moreover, modular networks have been

Manuscript received December 3, 1999; revised March 21, 2001. This work was supported by SeCyTUNC, CONICOR and CONICET.

L. Franco was with the Condensed Matter Group, Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, (5000), Córdoba, Argentina. He is now with the Cognitive Neuroscience Sector at Scuola Internazionale Superiore di Studi Avanzati (SISSA), Trieste 34014, Italy (e-mail: lfranco@sissa.it).

S. A. Cannas is with the Condensed Matter Group, Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Córdoba 5000, Argentina (e-mail: cannas@famaf.unc.edu.ar).

Publisher Item Identifier S 1045-9227(01)09511-X.

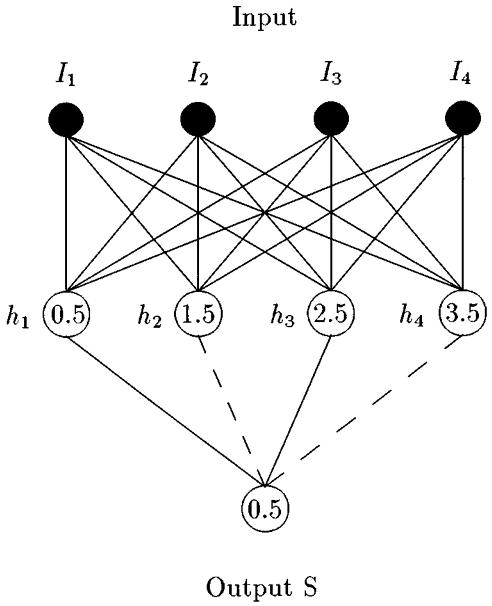

Fig. 1. Neural-network structure that computes the parity function of four input bits using the simplest architecture with only one fully connected hidden layer with four neurons. The threshold values are indicated inside the neurons while the synapsis values are 1 for the solid lines and $-1$ for dash lines.

used succesfully in several tasks as speaker recognition, face recognition, prediction of time series, etc. [1], [4], [15].

The family of networks and its general properties are presented in Section II.

In Section III, we analyze the generalization ability of the modular networks introduced in Section II, first by computing analytically the minimum number of examples needed for full generalization and second by numerical simulations. Both analytical and numerical results show that the generalization ability of these networks is systematically improved by the degree of modularity of the network. We also detect the existence of a phase transition from memorization to perfect learning (generalization) as the number of input bits is increased while the $f_{\max}$ is reduced, as first suggested by Patarnello and Carnevali [14].

Some conclusions and remarks are presented in Section IV.

## II. A Family of Architectures That Solve the Parity Problem

The simplest known solution (see [16]) for the $N$-bit parity function using linear threshold units has one hidden layer with a number of neurons equal to the number of inputs $N$. The network is fully connected and the value of the weights that solve the problem is very simple: all the weights connecting the input to the hidden layer are set to 1, the thresholds of the hidden neurons are the semi-integers numbers ranging from $1/2$ to $N - 1/2$, the connections from the hidden neurons to the output are alternatively set to 1 and $-1$ and the threshold output is set to $1/2$. From now on we will refer to this architecture as the "basic architecture." In Fig. 1, we show an example of this architecture for $N = 4$.

The network functions is as follows: when $i$ of the $N$ input neurons are ON, $i$ of the hidden neurons, those with thresholds less than $i$, are ON. Since the synaptic weights connecting hidden neurons to the output have alternate values 1 and $-1$, only when an odd number of inputs neurons are ON is the sum of the activated weights onto the output unit able to exceed its threshold of $1/2$ so as to activate the output neuron. An even number of active input neurons results in an even number of active hidden units, the sum of the activated weights onto the output unit is zero and the output unit remains inactive.

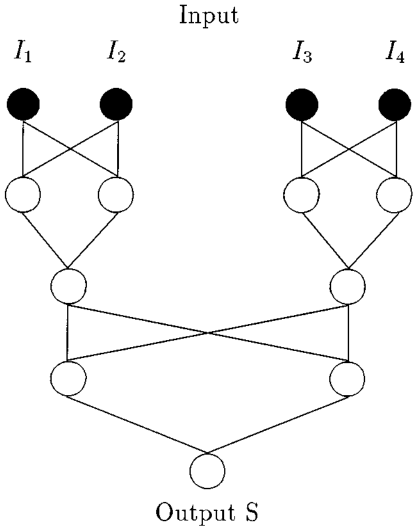

Fig. 2. Modular network structure to compute the parity function of four input bits with a $f_{\max} = 2$. The parity of pairs of input bits is computed and later the parity of the results is calculated using the same procedure.

We now introduce a family of architectures that generalize this previous basic architecture and exactly solves the parity function.

The architectures are constructed from a very simple and well-known property: the addition of two odd or even numbers gives an even number, while the addition of one odd and one even number is odd. The parity of an arbitrary number of bits can thus be computed by dividing them into groups and computing the parity of every group independently. This procedure can be repeated recursively until we obtain a single output result. The parity computation at every step can be performed by a neural network with the basic architecture. There are several options for grouping the bits at every step, each one of them associated with a particular architecture. One particular way is to build (when possible) all groups with the same number $m$ of bits. This choice generates a modular structure where the basic architecture that solves the parity of $m$ bits is repeated recursively.

Let us illustrate the idea with a simple $m = 2$ example. Consider for simplicity that $N$ is $2^k$ ($k > 0$); we then divide the input into $N/2$ pairs and compute the parity of every pair (an exclusive XOR function) with the well-known basic architecture (See Fig. 2). We obtain $N/2$ binary outputs that we group in $N/4$ pairs and so on. This architecture computes the parity of the $N$-bit input in $k = \log_2 N$ steps and involves a total number of $2k - 1$ hidden layers. An example for $N = 4$ is shown in Fig. 2.

The generalization to the case $N \neq 2^k$ is straightforward: every time we need to compute the parity of an odd number of bits, either at the input or in one of the intermediate steps, we leave one bit out compute the parity of the remaining bits with the previous procedure and include the remaining bit in a further computation. If we write $N = 2^k - l$ (with $l < 2^{k-1}$), the corresponding parity problem can thus been solved by an architecture with $2k - 1$ hidden layers. The structure will involve synapsis between neurons separated by more than one layer which can be visualized as mediated by subsequent neutral hidden neurons that just propagate their activity to the next layer without alteration. This structure has a fan-in max of $m = 2$.

This scenario can be easily generalized to the case $m > 2$ using the basic structure with $m$ bits. We now analyze the case where $N = m^k$; the $N \neq m^k$ case can be solved by the approach discussed in the preceding paragraph. In the case $N = m^k$ we obtain a modular structure that solves parity in $k = \log_m N$ steps, with $2k - 1$ hidden layers, a total number of neurons

$$N_n = 1 + 2m\frac{N-1}{m-1} \tag{2}$$

and a total number of synaptic weights

$$N_s = \frac{m(m+1)(N-1)}{m-1}. \tag{3}$$

These architectures have a $f_{\max} = m$. This parameter measures the degree of modularity of the networks; the case $m = N$ corresponds to the basic architecture with no modularity at all, while the minimum number $m = 2$ corresponds to maximum modularity.

Note that both $N_n$ and $N_s$ increase linearly with the number of inputs $N$. For large values of $m$, $N_n$ becomes almost independent of $m$ while $N_s$ increases linearly with $m$: in networks with a large number of inputs the number of synaptic weights can be reduced through small $m$ with only small increase in the total number of neurons. These considerations are important for hardware design.

## III. Generalization Ability

### A. Minimum Number of Examples for Full Generalization

To analyze the generalization ability of the networks presented in Section II we first calculate an upper bound for the minimum number of examples $M$ needed to obtain full generalization in learning the parity function. This number depends both on the target function (in this work we just study the parity function) and on the chosen architecture and it can be obtained analytically by analyzing directly the linear threshold equations derived by imposing the perfect learning of a set of examples. We introduced this method in a previous work [7] to analyze the influence of the selection of examples in the emergence of generalization ability for several networks with linear threshold units. In particular, we showed that perfect generalization can not be achieved with the basic architecture with continuous weights for the parity problem.

The idea of the method is as follows: the perfect learning of every one of the $2^N$ possible examples imposes a set of constraints on the synaptic weights. In the general case, the $2^N$ constraints will not be independent, but independent subsets exist. Since every constraint is associated with one particular example, the size of such subsets determines the minimum number of examples that ensures perfect generalization.

We now consider the case of "clipped" or "restricted" weights $J_{ij} = \{\pm 1\}$ and suppose that we use some appropriate learning algorithm that guarantees the zero error learning of the training examples (for instance, simulated annealing) starting from a random assignment of the weights.

We first analyze the basic architecture $N = m$. Let us consider the particular example with $N = m = 4$ showed in Fig. 1, where $I_i = \{0, 1\}$ $(i = 1, 2, 3, 4)$ are the input bits

$$h_i = \Theta\left[\sum_{j=1}^{4} J_{ij} I_j - T_i\right] \quad i = 1, 2, 3, 4 \tag{4}$$

determine the activity of the hidden neurons, $T_i$ are real thresholds and $J_{ij} = \pm 1$ are the input-to-hidden weights. The output $S$ follows from

$$S = \Theta\left[\sum_{k=1}^{4} w_k h_k - T\right] \tag{5}$$

where $T$ is also real and $w_k = \pm 1$ are the hidden-to-output weights.

Different choices of the thresholds $\{T_i, T\}$ allow for different solutions of the parity problem, every one associated with a different internal representation, i.e., a set of hidden unit activities $\{h_i\}$ for every input (see [16] and [6]). We choose the thresholds, as described in the previous section, $T = 0.5$ and $T_i = 0.5, 1.5, \ldots, N - 0.5$, for $i = 1, \ldots, N$. With this choice and clipped synaptic weights it can be seen (as it will become clear in our analysis) that there is only one possible internal representation except for trivial permutations of the hidden units. This property also holds for the more general case $0 \leq T < 1$ and $i - 1/2 \leq T_i < i + 1/2$. In order to simplify the analysis we will keep the thresholds to that values in all our calculations. We expect our results for the minimum number of examples needed for full generalization (MNEFG) to be independent of that constraint. Fixing the thresholds is equivalent to choose a particular internal representation for the solution of the target problem and the MNEFG is expected to scale in the same way with $N$ for any possible internal representation, as we verified for the parity problem in a fully connected network with continuous weights [7]. In the present problem we verified this assumption *a posteriori* by means of numerical simulations.

We will denote the examples by writing between square brackets the input values and the correct output separated by a colon. With our choice of threshold values the example [0000:0] is automatically solved by the network. Now consider the example [1000:1]. Since all thresholds are positive, $h_2 = h_3 = h_4 = 0$ and (5) with $S = 1$ requires $w_1 = h_1 = 1$. From (4) we, then obtain that $J_{11} = 1$. Perfect learning of all the examples with only one bit ON: [1000:1], [0100:1], [0010:1] and [0001:1] thus implies the following necessary conditions: $w_1 = J_{1j} = 1$, $j = 1, 2, 3, 4$.

We now consider the examples with two bits ON, for instance [1100:0]. From the previous conditions we have that $h_1 = 1$ and from (5) that $h_2 w_2 + 0.5 < 0$, which implies $w_2 = -1$ and $h_2 = 1$. Then from (4) we obtain that $J_{21} = J_{22} = 1$. Repeating this procedure with the rest of the examples with two bits ON we obtain that $J_{2j} = 1$, $j = 1, 2, 3, 4$, but notice that just two of the six possible examples are needed to ensure the fulfillment of the above conditions, provided that the active input bits do not overlap (for instance: [1100:0] and [0011:0]).

We now consider examples that contain three input bits ON; and two of the four available examples suffice to determine $w_3 = J_{3j} = 1$, $j = 1, 2, 3, 4$. Finally the single example with all the input bits ON, [1111:0] implies $w_4 = -1$ and $J_{4j} = 1$, $j = 1, 2, 3, 4$.

Hence, we see that for $N = 4$ it is enough to learn nine selected examples (slight above half of the total number $2^N = 16$ of examples) to obtain full generalization.

The generalization of this result to the case of $N$ input bits is straightforward: we analyze the complete set of $2^N$ examples in groups formed by the examples that have only $i$ bits ON and we take from every group the minimum number needed to ensure that every input bit appears at least once in the ON state. The total number of examples needed to obtain full generalization is thus given by

$$M = \sum_{i=1}^{N} \mathrm{Int}_+ \left(\frac{N}{i}\right) \tag{6}$$

where $\mathrm{Int}_+(x)$ equals $x$ if $x$ is an integer and it equals the closest integer greater than $x$ if $x$ is real. For large values $N \gg 1$, this number scales as $M \propto N \log(N)$, quite smaller than both the total number of examples $2^N$ and the total number of synapsis $N^2 + N$.

We now analyze the case $m < N$.

Let us start with a particular case with $N = 9$, $m = 3$. This network has a modular structure derived from the basic architecture with $N = m = 3$. The thresholds in every module are fixed as before.

As we explained in the previous section, the three input modules have to compute independently the parity of the corresponding input bits, while the output module has to compute the parity of the previous results. Hence, a natural choice is to start with all the examples needed to learn the parity in every one of the three input modules, that is, examples containing different combinations of zeros and ones in the input of one particular module and zero in the rest of them (for instance, [xxx 000 000:y], where [xxx:y] is one of the examples needed for a basic architecture with $N = m = 3$). In this case the outputs of the input modules will have at most one bit ON. Therefore, this set of $3\sum_{i=1}^{3} \mathrm{Int}_+ (3/i) = 18$ examples will fix all the synaptic weights associated with the first hidden neuron of the output module (that is, the neuron with threshold $1/2$) together with all the internal synaptic weights of the input modules.

Once this learning task is performed, every input module acts as a single unit, as far as the output module is concerned. Hence, we need another set of $\sum_{i=2}^{3} \mathrm{Int}_+ (3/i) = 3$ appropriated chosen examples in order to fix the rest of the synaptic weights of the basic output architecture. These examples have to contain two of them giving different combinations of module outputs with two bits ON (for instance [111 111 000:0] and [111 000 111:0]) and the example with all the input bits ON. Then, we see that a number $M = 21$ over $2^9 = 512$ possible examples is enough to learn the parity problem. This number has to be compared with $M = 29$ derived from (6) for a basic architecture with $N = m = 9$.

The generalization of the argument to the general case $N = m^k$ is straightforward. The number of examples needed to learn the parity in the $m^{k-1}$ input modules is

$$m^{k-1} \sum_{i=1}^{m} \mathrm{Int}_+ \left(\frac{m}{i}\right) = N + m^{k-1} \sum_{i=2}^{m} \mathrm{Int}_+ \left(\frac{m}{i}\right).$$

Once this learning task is performed, the input modules act as single units for the next layer of $m^{k-2}$ modules. For this layer, all the examples with only one input bit ON have already been learned. We only needed another set of

$$m^{k-2} \sum_{i=2}^{m} \mathrm{Int}_+ \left(\frac{m}{i}\right)$$

appropriately chosen examples to fix the rest of the synaptic weights of this layer. Repeating this procedure for the remaining $k - 2$ layers of modules leads to

$$\begin{aligned} M(N) =& N + \left[\sum_{j=0}^{k-1} m^j\right] \left[\sum_{i=2}^{m} \mathrm{Int}_+ \left(\frac{m}{i}\right)\right] \\ =& N + \frac{N-1}{m-1} \sum_{i=2}^{m} \mathrm{Int}_+ \left(\frac{m}{i}\right) \end{aligned} \tag{7}$$

for $N = m^k$. Notice that for large values of $N \gg 1$ and strong modularity $m \ll N$, $M(N)$ scales linearly with $N$ instead of the $N \log(N)$ scaling for $m \sim N$.

The analysis is more complex for $N = m^k - l$, with $l < m^{k-1}$. This case can be solved by constructing a modular network with $m^k$ input units and training it with examples where the extra $l$ input bits are always set to zero. An upper bound to the number of examples needed to train this network is given by $M(m^k - l) \leq M(m^k)$, where $M(m^k)$ is given by (7) and the equality holds for $l = 0$.

Consider now the case of *continuous weights* $J_{ij}$. Again, the modular structure of the networks allows us to carry out the analysis recursively from the properties of the constituting modules.

For continuous weights the restrictions imposed by the learning of $K$ examples appear in the form of a set of $K$ simultaneous inequalities. In a previous work [7] we showed that, for a basic architecture of $m$ input bits with the choice of the thresholds adopted here, full generalization implies the fulfillment of $2^m - 1$ *independent* inequalities: only the learning of the whole set of possible examples (except for the trivial one with all the input bits set to zero, which is automatically fulfilled) ensures full generalization with this architecture.

We now repeat the procedure used in the case of clipped synaptic weights for $N = m^k$ and $m < N$. In a first step we teach the network all the $N$ examples with only one bit . This lead to a set of independent inequalities for all the weights associated with hidden neurons with threshold equal to $1/2$. To obtain the inequalities related to the rest of the weights of the input modules, we have to teach in a second step all the

TABLE I
SOME FEATURES OF THE NETWORKS USED TO COMPUTE THE $N$-BIT PARITY FUNCTION WITH A $f_{\max} = m$. SEE TEXT FOR THE DEFINITION OF THE $\mathrm{Int}_{+}(x)$ FUNCTION

| Number of neurons | Number of synapses | Depth | Examples needed for generalization | |
|---|---|---|---|---|
| | | | Restricted weights | Non-Restricted weights |
| $1 + 2m\frac{N-1}{m-1}$ | $\frac{m(m+1)(N-1)}{m-1}$ | $2\log_m N$ | $N + \frac{(N-1)}{m-1}\sum_{i=2}^{m} Int_{+}\frac{m}{i}$<br>$\sim \mathcal{O}(N)$ for $m \ll N$<br>$\sim \mathcal{O}(NlogN)$ for $m \sim N$ | $N + \frac{N-1}{m-1}(2^m - m - 1)$<br>$\sim \mathcal{O}(N)$ for $m \ll N$<br>$\sim \mathcal{O}(2^N)$ for $m \sim N$ |

$m^{k-1}\sum_{i=2}^{m}\binom{m}{i}$ examples containing more than one bit ON in every of the input modules and zero in the rest of them ($\binom{m}{i}$ being the binomial coefficient). But, for the successive layers of modules, all the inequalities associated with just one input bit ON in every module have already been set in the first step. Hence, to impose the rest of the synaptic weights of the layer $j$ the appropriated inequalities we need only another set of $m^{k-j}\sum_{i=2}^{m}\binom{m}{i}$ examples. Then, the total number of examples is

$$\begin{aligned} M(N) =& N + \frac{N-1}{m-1}\sum_{i=2}^{m}\binom{m}{i} \\ =& N + \frac{N-1}{m-1}(2^m - m - 1) \end{aligned} \qquad (8)$$

for a modular network with continuous weights and $N = m^k$. We note that it is the same result obtained for the clipped case, (7) but changing the term $\mathrm{Int}_{+}(m/i)$ by $\binom{m}{i}$. For $m = N$ the result $M = 2^m - 1$ is recovered. As before, $M(N)$ in (8) gives an upper bound for the case $N = m^k - l$. Note that for $m \ll N$ and $N \gg 1$, $M(N)$ scales linearly with $N$ for continuous weights. The features obtained for the modular architectures are summarized in Table I.

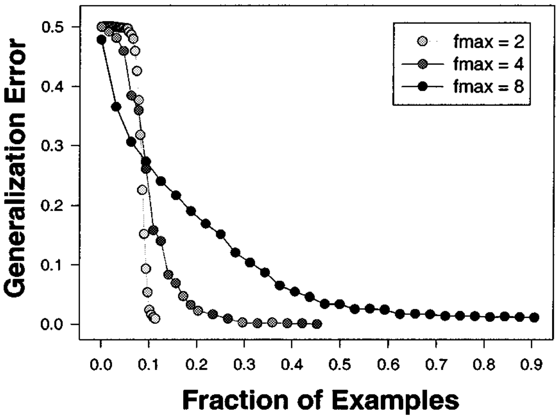

Fig. 3. Generalization error versus fraction of random selected examples for three networks implementing the parity function of eight bits using different architectures. In dark the results correspond to the case of a single hidden architecture with $f_{\max} = 8$, in dark gray those corresponding to an architecture with three hidden layer with a $f_{\max} = 4$ and in light gray a five-hidden layer architecture with $f_{\max} = 2$.

### B. Numerical Results

We performed numerical simulations for $N = 8$ networks with clipped weights, using simulated annealing as the learning algorithm and parity as the target function. The parameters of the simulated annealing algorithm (i.e., initial temperature, rates of temperature decay, etc.) were kept constant in all simulations. The threshold values were fixed to those mentioned in the previous section. We performed several *a posteriori* checks without fixing the thresholds obtaining similar results; however, a slower convergence rate of the algorithms was obtained for the case of variable thresholds.

In order to investigate the effect of the modularity we compared three different networks with $f_{\max} = 2$ ($m = 2$, maximum modularity), $f_{\max} = 8$ ($m = N = 8$, basic architecture) and the intermediate case $f_{\max} = 4$. The last case consists of a mixed architecture with two $m = 4$ input modules and an $m = 2$ output module. We calculated the learning curves: the average generalization error $\epsilon_g$ *versus* the fraction $\rho_e = N_e/2^N$, $N_e$ being the number of examples used in the training. The results were averaged over different sets of $N_e$ examples and over different initial realizations of random weights $\{J_{ij} = \pm 1\}$. Typical sample sizes run from 50 to 100. The training for every initial condition and set of examples was continued until zero learning error was achieved. The generalization error was then calculated over the whole set of $2^N$ examples.

In Fig. 3, we compare the learning curves for the three networks. For $f_{\max} = 8$, the generalization error $\epsilon_g$ decays very slowly and approaches $\epsilon_g = 0$ asymptotically as $\rho_e$ approaches to 1, showing a lack of generalization that it is sometimes encountered in neural-network applications. A systematic improvement is observed as the modularity is increased. A dramatic change of behavior occurs for $f_{\max} = 2$, where $\epsilon_g$ remains almost constant $\epsilon_g \sim 0.5$ for $N_e < M$ and decays suddenly to zero with a few more examples, suggesting a phase transition from memorization to perfect learning for large lattices. This effect has been pointed out before by Patarnello and Carnevali [14], for neural networks composed of Boolean gates with $f_{\max}$ of two. This transition seems to be an effect associated to the extreme modularity. For $f_{\max} = 4$ the $\epsilon_g$ decays smoothly to zero.

To gain an insight about the relationship between the above mentioned transition and the modularity we analyzed how the learning procedure affects the different modules as we increase $\rho_e$ for $f_{\max} = 2$. For $N = 8$ the resulting network has three

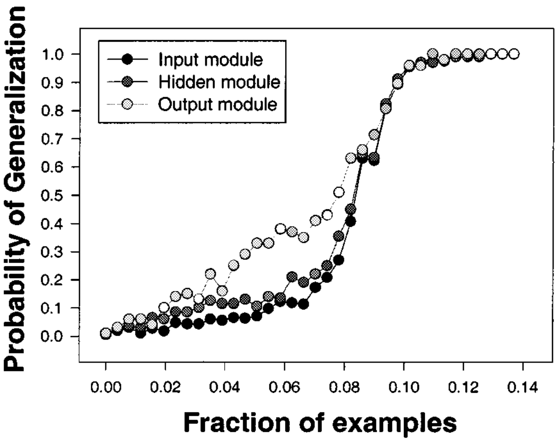

Fig. 4. Probability of generalization versus fraction of examples calculated for the different modules (input, hidden, and output) of a modular network with $N = 8$ and $f_{\max} = 2$.

layers of modules: input, intermediate and output. We calculate the probability of learning correct synaptic weights in a given module which implement the target function (remember that, since the thresholds are fixed there is just one correct set of weights). The numerical calculations were carried out over samples of size 1000. We first verified that these probabilities are the same for all modules in the same layer, as expected from the symmetry of the network. In Fig. 4, we compare such probabilities versus $\rho_e$ for modules in different layers. We see that learning occurs from bottom to top: as we increase $\rho_e$ the learning probabilities for the input and hidden modules remain at very low values ($\sim$ 10%) while the probability for the output module steadily increases. When $\rho_e$ reaches the value $\rho_e \sim 0.06 \sim M/2^N$, where the output probability is around 40%, the input and hidden probabilities experience a sudden growth, they become very similar and converge rapidly to one.

In order to establish the practical advantage of modular networks, we measure the average CPU time needed to learn a number $N_e$ of examples needed to achieve $\epsilon_g \leq 0.05$. More precisely, we run simulations with fixed $N_e$ up to a predetermined maximum number of simulated annealing iterations. We repeat this procedure starting from different initial conditions and for different sets of examples of size $N_e$. In many runs the learning is not successful: the algorithm does not converge to zero learning error. We compute the total CPU time (in ms) needed to obtain $K$ successful learning sessions (including the CPU time wasted in the cases where the network does not learn) and divide it by $K$ ($K = 50$ was enough to stabilize the average).

In Fig. 5 we compare the average CPU time for the three modular networks considered here for $N = 8$, for both selected and random training sets. While for the former the improvement is very impressive and monotonic as $f_{\max}$ decreases, for the latter we see that the high generalization ability for $f_{\max} = 2$ is obtained with very poor efficiency (note the logarithmic scale in the ordinates of the plot), but the CPU time does not increase monotonically as $f_{\max}$ increases and is optimal $f_{\max} = 4$.

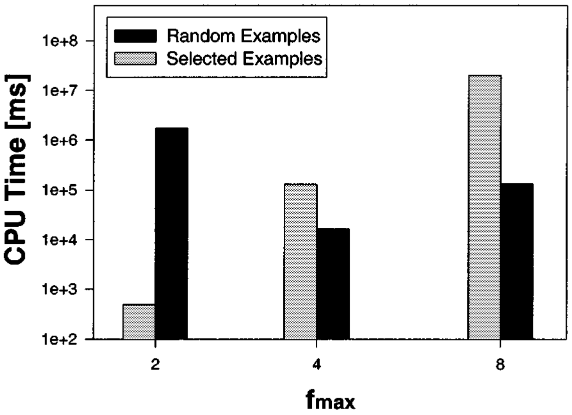

Fig. 5. Learning CPU time for the parity function using three different architectures with $f_{\max} = 2, 4$ and eight using random and selected training sets.

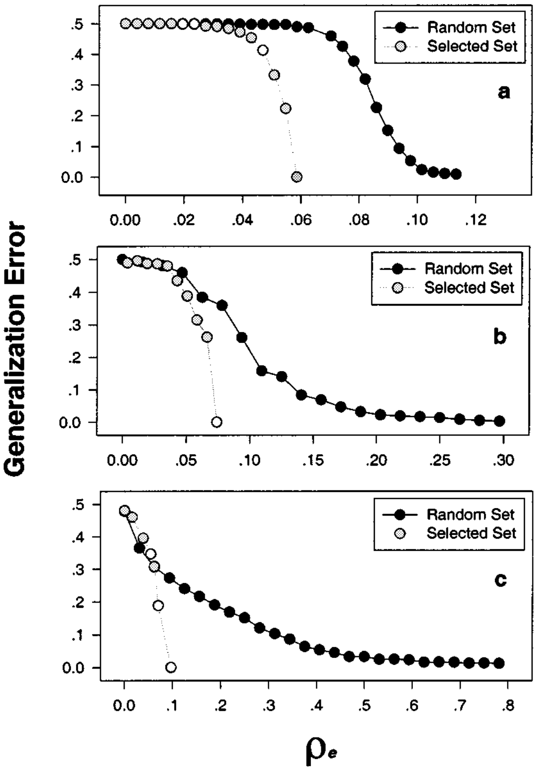

Fig. 6. Generalization error versus fraction of examples in the training set for random and selected examples for networks solving the parity funcion with $N = 8$ and $f_{\max} = 2, 4$ and eight.

Next, we analyzed the influence of example selection on the learning ability of the modular networks. In Fig. 6 we compare $\epsilon_g$ versus $N_e/2^N$ ($N = 8$) for a random choice of the training set over all the possible examples and a random selection *only* over the subset that ensures full generalization (see the preceding section). Results for $f_{\max} = 2, 4$ and eight are shown in Fig. 6(a)–(c), respectively. We see that for $f_{\max} = 2$ the influence of modularity is so strong that little is gained from example selection. On the other hand, we observe that example selection can lead to a remarkable improvement in the generalization ability of nonmodular networks ($f_{\max} = N = 8$) but, at the cost of poor efficiency (see Fig. 5).

Finally, we analyze how the minimum number of examples needed for full generalization $M$ scales with the number of synapses $N_s$. From Table I we see that for modular networks

($m \ll N$) $M$ scales linearly with $N_s$, $M \sim \alpha(m) N_s$, both for restricted and continuous weights. In the first case $\alpha(m)$ decreases slowly with $m$. This linear behavior has been observed in the number of *random* examples needed for generalization in fully connected networks [9]. On the other hand for nonmodular networks ($m \sim N$) very different scaling behaviors are observed. For restricted weights we see that $M/N_s \equiv \ln(N_s/2N_s^{1/2})$ while for continuous weights we obtain $M \sim 2^{N_s^{1/2}}$.

## IV. Conclusion and Discussion

We have constructed a family of architectures capable of implement the parity function in a simple way, either with restricted (discrete) or continuous synaptic weights. We showed that several properties of these networks, like depth, number of synapses, generalization ability, etc., are controlled by a single parameter, namely, the fan-in max ($f_{\max}$). We analyzed, both analytically and numerically, several of these properties as $f_{\max}$ is varied, comparing the different behaviors observed in networks with restricted and continuous weights.

The study of learning properties of the parity function in different architectures is of importance by several reasons. First of all, being it a standard for testing learning algorithms, it is always of interest to know general properties of different types of implementations. Conversely, knowing how well a given architecture deals with the parity may serve as a benchmark for the learning capability of the network. In other words, being the parity one of the most difficult learning tasks, we may think its learning properties as "bounds" for the learning properties of "easier" functions. Hence, it may also serve for comparing learning performances of different architectures. In this sense, the present study indicates that generalization can be highly improved as the degree of modularity of the network increases. Moreover, we found that a high degree of modularity can lead to a phase transition from memorization to perfect learning. This transition occurs when the fraction of training examples reaches the minimum number of examples needed for full generalization $M$. This result supports our previous proposal (see [7]) of $M$ as a combined measure of target function and architecture complexities. One standard measure (function independent) of the learning capacity of an architecture is the Vapnik–Chervonenkis (VC) dimension (see, for example, [9], for its definition). Generalization starts at the VC dimension, that is, no generalization at all is possible if the number of training examples is below it, for any target function. Now, for a particular target function, perfect generalization is only possible is the number of training examples exceeds $M$, which of course is always greater than the VC dimension. Numerical evidence have shown that the generalization properties seem to be directly related to the scaling properties of $M$ with the number of input neurons $N$ ([7]). The present work shows that generalization becomes optimal in the learning of the parity using a random sampling of examples with an extremely modular network $m = 2$ (at least as far as the generalization error is concerned), since $\epsilon_g \approx 0$ when $\rho_e \equiv M/2^N$. This result should be compared with the efficiency in terms of the average CPU time wasted by the learning algorithm to find a solution with a given generalization error and fixed number of training examples by the different architectures. Our results show that the very good performance for the generalization error in the case $m = 2$ is obtained at the cost of a very poor efficiency (see Fig. 5). Moreover, the minimum observed in the CPU time for $m = 4$ suggests that an adequate tradeoff between generalization and efficiency can be obtained by using modular structures with intermediate values of $f_{\max}$.

We have also shown that for structures with a high degree of modularity ($m \ll N$) the ratio between $M$ and the number of synapses $M/N_s$ is constant both for restricted and continuous weights. This linear scaling of $M$ with $N_s$ is always of interest to hardware design. On the other hand, for networks with a low degree of modularity ($m \sim N$) very different scaling behaviors are observed, depending on the type of weights. For continuous weights the ratio $M/N_s$ *increases exponentially* with $N_s$, while for restricted weights such ratio *decreases* with $N_s$. Although this result seems to indicate the general convenience of using restricted instead of continuous weights, care must be taken concerning the computability (that is, the number of different target functions that the architecture is capable of implement) of every type of network. This result may be very particular of the parity; in general it is expected a much more lower computability using discrete instead of continuous weights. Of course the question of computability arises for all the results here presented. However, we believe that, though more restricted than fully connected networks, the modular structures here presented are capable to implement a great variety of target functions, at least those sharing with the parity what we can call the "self-similarity" property: that is, problems which can be divided into subproblems similar to the original one. Examples of these kind of problems are the arithmetic ones: addition, subtraction, multiplication, etc.

## Acknowledgment

The authors acknowledge S. Solla for fruitful discussions and for a critical reading of the manuscript.

## References

[1] Y. Bennani, "Multiexpert and hybrid connectionist approach for pattern recognition: Speaker identification task," *Int. J. Neural Syst.*, vol. 5, pp. 207–216, 1994.

[2] P. Carnevali and S. Patarnello, "Exhaustive thermodynamical analysis of Boolean learning networks," *Europhys. Lett.*, vol. 4, no. 10, pp. 1199–1204, 1987.

[3] K. Chen, L. Yang, X. Yu, and H. Chi, "A self-generating modular neural-network architecture for supervised learning," *Neurocomput.*, vol. 16, pp. 33–48, 1997.

[4] M. N. Dailey and G. W. Cottrell, "Organization of face and object recognition in modular neural-network models," *Neural Networks*, vol. 12, pp. 1053–1074, 1999.

[5] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield, "Large automatic learning, rule extraction and generalization," *Complex Syst.*, vol. 1, pp. 877–922, 1987.

[6] L. Franco and S. A. Cannas, "Solving arithmetic problems using feed-forward neural networks," *Neurocomput.*, vol. 18, pp. 61–79, 1998.

[7] ——, "Generalization and selection of examples in feedforward neural networks," *Neural Comput.*, vol. 12, pp. 2405–2426, 2000.

[8] H. Fung and L. K. Li, "Minimal feedforward parity networks using threshold gates," *Neural Comput.*, vol. 13, pp. 319–326, 2001.

[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: McMillan, 1994.

[10] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1991.

[11] R. Impagliazzo, R. Paturi, and M. E. Saks, “Size-depth tradeoffs for threshold circuits,” *SIAM J. Comput.*, vol. 26, no. 3, pp. 693–707, 1997.

[12] M. L. Minsky and S. A. Papert, *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[13] I. Parberry, “Circuit complexity and feedforward neural networks,” in *Mathematical Perspectives on Neural Networks*, P. Smolensky, M. Mozer, and D. Rumelhart, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1996, pp. 85–111.

[14] S. Patarnello and P. Carnevali, “Learning networks of neutrons with boolean logic,” *Europhys. Lett.*, vol. 4, pp. 503–508, 1987.

[15] V. Petridis and A. Kehagias, *Predictive Modular Neural Networks: Applications to Time Series*. Boston, MA: Kluwer, 1998.

[16] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, vol. 1.

[17] G. Tesauro and R. Janssens, “Scaling relationships in backpropagation learning: Dependence on predicate order,” Center Complex Syst. Res., Urbana-Champaign, IL, Tech. Rep. CCSR-88-1, 1988.

[18] C. V. den Broeck and R. Kawai, “Learning in feedforward Boolean networks,” *Phys. Rev. A*, vol. 42, no. 10, pp. 6210–6218, 1990.

**Leonardo Franco** received the M.Sc. degree in physics in 1995 and the Ph.D. degree in 2000 with a dissertation on generalization properties of feedforward neural networks, both from Córdoba National University, Córdoba, Argentina.

In September 2000, he joined the Cognitive Neuroscience Sector at SISSA, Italy, as a Postdoctoral Fellow, where he is working on computational neuroscience. His research interests include generalization properties and complexity of neural networks, information theory, face recognition, and modeling of the visual system.

**Sergio Alejandro Cannas** was born in Córdoba, Argentina, on November 21, 1961. He received the M.Sc. degree from the National University of Córdoba in 1984 and the Ph.D. degree from the Centro Brasileiro de Pesquísas Físicas, Brazil, in 1992, both with a thesis and dissertation on statistical physics.

Since 1994, he has been an Assistant Professor of the National University of Córdoba. In 1995, he became a Researcher of the National Research Council (CONICET) of Argentina. His research interests include neural networks and statistical physics of complex systems.