Analysis of Fault Tolerance in Artificial Neural Networks

Vincenzo Piuri

Department of Electronics, Politecnico di Milano, Piazza L. da Vinci 32, 20133 Milan, Italy

Received March 17, 1997; revised May 25, 2000; accepted June 19, 2000

Wide attention was recently given to the problem of fault-tolerance in neural networks; while most authors dealt with aspects related to specific VLSI implementations, attention was also given to the intrinsic capacity of survival to faults characterizing the neural modes. The present paper tackles this second theme, considering in particular multilayered feed forward nets. One of the main goals is to identify the real influence of faults on the neural computation in order to show that neural paradigms cannot be considered intrinsically fault tolerant (i.e., able to survive to faults, even several of the most common and simple ones). A high abstraction level (corresponding to the neural graphs) is taken as the basis of the study and a corresponding error model is introduced. The effects of such errors induced by faults are analytically derived to verify the probability of intrinsic masking in the final neural outputs. Then, conditions allowing for complete compensation of the errors induced by faults through weight adjustment are evaluated to test the masking abilities of the network. The designer of a neural architecture should perform such a mathematical analysis to check the actual fault-tolerance features of his or her system. Unfortunately, this involves a very high computational overhead. As a cost-effective alternative for the designer, the use of a behavioral simulation is proposed for a quantitative evaluation of the error effect on the neural computation. Repeated learning (i.e., a new application of the learning procedure on the faulty network) is then experimented to induce error masking. Experimental results prove that even single errors affect the computation in a relevant way and that weight redistribution is not able to induce complete masking after a fault occurred, i.e., the network cannot be considered per se intrinsically fault tolerant and it is not possible to rely on learning only in order to achieve complete masking abilities. Mapping criteria of physical faults onto the abstract errors are finally examined to show the usability of the proposed analysis in evaluating the actual robustness of a neural networks' implementation and in identifying the critical areas where architectural redundancy should be introduced to achieve fault tolerance. © 2001 Academic Press

1. INTRODUCTION

Artificial neural networks (ANNs) offer an attractive solution for complex non-linear problems in many critical application areas (signal and image processing, real-time control, etc.) when an algorithmic approach cannot be easily defined. On the other hand, advances in integration technologies and architectural design now allow implementations at reasonable costs: in fact, even commercial systems are by now available to large classes of users.

The possibility of VLSI or WSI implementation of ANNs and their application in mission-critical areas (e.g., in aerospace environments and in critical control systems) makes defect and fault tolerance such an important issue in system design that it should be taken into account even from the initial design stages at the behavioral level. Since computation and information are not localized in the ANN but are holographically distributed, an error in a single neuron or in a synaptic weight affects the whole computation, although—in general—it does not completely destroy any part of it. As a consequence, the concept of error confinement, basic to most conventional fault-tolerance policies, cannot be applied in its usual way to limit the error propagation [1]: approaches tailored to the neural paradigms must therefore be envisioned in order to achieve the best performance at limited costs.

Several authors dealt with different aspects of defect and fault tolerance in relation to specific implementations of neural nets. In [2], behavioral fault models are defined with reference to multilayered nets implemented by analog systems, and the impact of the physical faults on the operation of the single device is examined at various abstraction levels. In [3], a relationship is identified between the physical defects and failures and the maximum size of the device for a given analog implementation approach. In [4], a digital array architecture supporting the mapping of a number of neural nets is envisioned: solutions for defect and fault tolerance are discussed, with reference to the supporting architecture (and to the mapping technique) rather than to the characteristics of the neural nets mapped onto it. In [5, 6], testing is treated at the behavioral level with specific reference to neural paradigms in order to verify the testability of such computing structures and to generate efficiently the related test procedures for end-of-production and offline verification of digital implementations. In [7–12], concurrent error detection techniques at architectural and functional levels were proposed to certify the correctness of each individual result and, possibly, correct the errors.

On the other hand, several studies were also performed to introduce fault tolerance directly into the neural paradigms, independent from their implementation. In [13, 14], the effects of errors in the behavior of neural operators on the whole neural computation were analyzed at the behavioral level by deriving the probability that neurons' and networks' outputs are correct but without any reference both to a general methodology that could be used by a designer in the practice and to the relationships between faults and errors. Besides, these techniques are able neither to certify the actual correctness of the individual network result nor to forecast the exhaustion of the intrinsic error masking ability, allowing the use of possibly-erroneous results. In [15, 16], approaches to increase the error masking ability were presented by modifying the learning procedure in order to force a higher insensitivity to errors; this can be achieved by forcing the final working points of the neurons toward the saturation regions of the nonlinear activation functions, so that even a large variation of the weighted summation affects the neuron output either marginally or not at all. This technique can be seen as somehow corresponding to the information redundancy approach for classical digital networks [17]. However, this kind of masking ability does not cover many relevant error classes or is not present in several neural paradigms. The error confinement requirement was relaxed by requiring simply that the information provided by a faulty unit be kept consistent with a given error assumption (e.g., requiring that the output of a neuron be kept fixed to a given value as soon as the neuron is found to be faulty). In [18], correlation between generalization and fault tolerance was empirically studied for the linear programming algorithm by injecting errors during learning. In [19], a very interesting analysis of the fault-tolerance characteristics in the backpropagation model was presented. The generalization ability is exploited in a highly efficient way to enhance the intrinsic masking and to incorporate—in the network configuration—the ability to deal with a given amount of error due to possible faults; this is achieved during learning by adopting an error function which is a linear combination of the criterion measuring the difference between the faulty and the fault-free networks and the traditional error function of the back-propagation algorithm. In [20], partitioning and distribution of a configured ANN among a larger number of neurons is shown to limit the individual neuron's contribution to the final result so that the fault-free neurons are supposed to be able to generate the correct (or near correct) final result; this requires so many neurons that any hardware implementation becomes infeasible. It is important to point out that all the above techniques assume the errors to be small: unfortunately, this is not generally true, especially for digital implementations [21].

Instead of creating robust architectures without any concern to the specific kind of computation performed or robust neural paradigm without reference to the underlying architecture, an alternative comprehensive approach consists of:

— analyzing the intrinsic robustness (whether and within which bounds) of the neural paradigm for the given behavioral error model (related to the abstract neural model), by evaluating the impact of such errors on the neural computation in a technology-independent way,

— mapping the actual physical faults onto the behavioral model for the envisioned implementation architecture and technology, i.e., transforming the physical fault probabilistic distribution into the implementation-specific probabilistic distribution of the behavioral errors,

— and evaluating the implementation's robustness by identifying the critical architectural areas (if any) with respect to the implementation-specific error distribution; in such areas ad-hoc fault-tolerance policies (e.g., [4, 7–12]) must be introduced at the architectural level to provide the desired level of fault tolerance.

The present paper adopts such an approach to introduce a structured design methodology, which takes into account both the partial ability of intrinsic masking of the neural paradigms and the need of architectural support to overcome the intrinsic inability of complete error masking. By exploiting both the intrinsic fault tolerance ability of the neural paradigms and the necessary architectural support enhancing such an ability, it is possible to achieve an implementation having high fault tolerance at limited costs. Architectural features are in fact added only when the neural paradigm is not sufficient to achieve the desired protection. Some preliminary ideas were presented in [21, 22].

The basic goal of the paper is to show that the neural networks cannot be considered *completely* fault tolerant by relying on the intrinsic (even if enhanced) masking ability only. The analysis method followed to such a purpose can be adopted as a guideline by the designer in order to identify the critical parts of his or her neural implementation. The designer's attention will then be focused on such areas as introducing the suitable solutions for architectural-level fault tolerance.

As an example, the multilayered back-propagation networks is chosen in this paper, according to the classical, completely-connected model. An initially error-free network is assumed, having completed a nominal supervised learning phase on the required set of input pattern classes when the null-error condition is reached (being this is consistent with the behavioral approach, by which only the system's response at its outputs can be observed and evaluated).

The behavioral error model is first introduced in Section 2 by taking into account both single and multiple errors due to faults. A theoretical analysis of the errors' influence on the neural computation is then presented. While most of our treatment is performed on the single-error assumption, the extension to multiple errors is straightforward. Quantitative evaluations cannot be easily obtained in general terms, depending not only on the errors' magnitude but also on the characteristics of the nonlinear evaluation functions. To overcome this limitation, extensive simulations were carried out for a set of sample networks with varying numbers of neurons and different distributions of neurons among layers, all designed to operate on the same set of input classes. Results of such simulations, presented in Section 4, prove that even single errors affect the computation in a relevant way, i.e., in general the neural paradigms cannot be considered per se intrinsically *completely* fault tolerant.

Having determined the effects induced by errors, the second step toward assessment of fault-tolerance consists of verifying whether it is possible to overcome the presence of errors by modifying the variable parameters present in the network (i.e., the synaptic weights), in other words, if weight redistribution (i.e., learning) alone is able to provide *complete* masking ability. To this end, a simple form of error confinement can be preliminarily adopted (e.g., by setting to zero a faulty synaptic weight or by setting to a fixed and known value the output of the fault-affected neuron). Then, two alternative approaches can be explored to grant that the deviation of the networks' outputs with respect to the expected ones is either null or kept within predetermined bounds. In the first, simplest, case, it can be requested that the inputs to the evaluation functions of all neurons in the layer immediately following the fault be kept identical to the expected ones in the error-free net. If a solution of this equation's system exists, it is possible to guarantee that learning in the presence of the faulty component can provide error masking (actually, direct application of the solution's weights obtains the same result). While the analytical relations thus derived are relatively simple (nonlinearities are excluded), the simplification of the problem is such that arguably many cases in which fault-tolerance could still be achieved would not be solved in this way. In the most general case, propagation of errors and of weight variations through all layers following the fault should be taken into account; nonlinearities make this approach extremely complex, although in the assumption of small error-induced deviations the system could be linearized.

The analytical conditions which allow us to state whether a given network, designed to operate on a given set of input classes, can be made to operate correctly even in the presence of one (or more) errors are discussed in Section 3. Actually, the mathematical complexity of the relations obtained is such that a numerical solution would not be feasible, apart from very simple networks; weight modification and redistribution can be achieved by repeating the learning phase on the error-affected network and (obviously) with the nominal set of input classes. Simulation results obtained by this relearning technique are presented in Section 4. They show that the presence of faults is not *completely* masked by the characteristics of the neural paradigm alone (even if with enhanced fault tolerance ability): some residue error is still present and, in the general case, is not negligible, even if the influence on the network's outputs is reduced with respect to the situation before weight adjustment.

After having identified until which point learning can be exploited to introduce effectively error masking, the robustness evaluation for the envisioned implementation must be based on mapping the physical faults affecting specific implementations onto the behavioral errors of the abstract model. Examples of this analysis are presented in Section 5. This allows the designer to transform the physical fault probability distribution into the actual probability distribution of the behavioral errors. In turn, this proved information about the criticality of the fault classes whenever they are not intrinsically masked by the neural computation, they frequently occur, and they affect the neural results in a nonnegligible way. Exploitation of all this information for enhancing the neural implementation design is discussed in Section 6: the designer can in fact identify the critical components which are not protected enough by the intrinsic masking of the neural computation and adopt the suitable architectural strategies to complete the protection.

2. THE ERROR MODEL AND ITS INFLUENCE ON THE NEURAL COMPUTATION

The network model adopted in this paper is the classical multilayered perceptron proposed by Rosenblatt (see [23]), in which the operation of any single neuron n_k is given by:

$$x_k = f\left(\sum_j w_{kj} x_j - \theta_k\right). \tag{1}$$

The individual neuron can thus be modeled as in Fig. 1, where each synapsis connecting neuron j of layer $i-1$ with neuron k in layer i is associated with the related

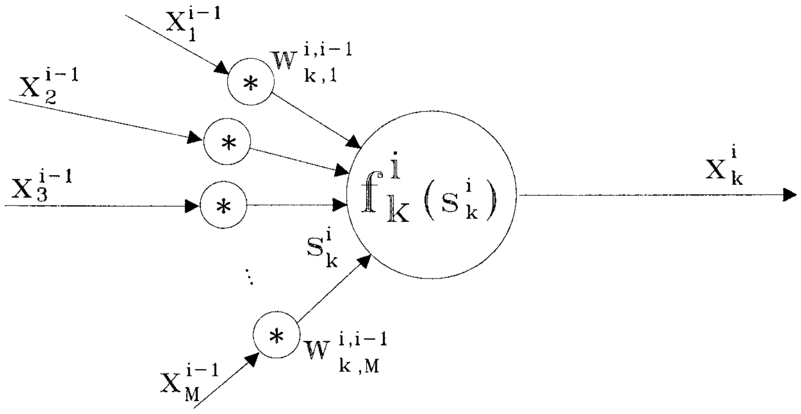

FIG. 1. Neuron's model.

synaptic weight $w_{kj}^{i,i-1}$ and the operator multiplying such a weight by signal x_j^{i-1} produced by neuron n_j^{i-1}. The summation over all input signals to n_k^i and the subsequent neural nonlinear evaluation function are represented by individual corresponding operators associated with neuron n_k^i.

Upon this behavioral neuron model, a behavioral error model can be defined depending only on the behavior of the neuron itself and independent of technological implementations. More specifically, in the behavioral neuron model, the following errors may occur:

a. unexpected value of input signal x_j^{i-1}: in a physical implementation, this can be due to a fault internal to n_j^{i-1}, to faulty interconnections, or even to external noise affecting the system;

b. errors affecting the synaptic weight $w_{kj}^{i,i-1}$ or the associated multiplication by x_j^{i-1}: in the absence of any implementation detail, they are to be considered as indistinguishable;

c. errors affecting the summation inside the neuron;

d. errors affecting the nonlinear evaluation function; actually, these errors lead to the creation of an unexpected value on x_k^i and thus are equivalent to errors of the class *a*.

Let us now consider the sensitivity of the neural net to such various errors. Generality leads us to take into account continuous signals, and, in particular, continuous nonlinear evaluation functions; the particular case of two-state signals and of step functions can be derived easily.

The effects of errors modifying the input to the evaluation function of neuron n_j are first analyzed. As a reference, the two most meaningful instances of the evaluation function (namely, the sigmoid and the step function) are considered.

In the first case, the sigmoid for neuron n_k can be expressed as

$$f_k(s_k) = \frac{1 - e^{s_k}}{1 + e^{-s_k}}, \tag{2}$$

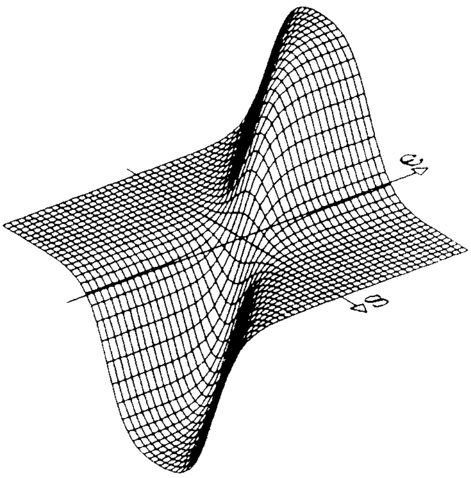

FIG. 2. Local error for the sigmoid function.

where $s_k = \sum_j w_{kj} x_j - \theta_k$. The injection of an error ε in s_k produces an error in the output given by:

$$e_k(s_k, \varepsilon) = f_k(s_k + \varepsilon) - f_k(s_k) = 2\,\frac{e^{-s_k}(1 - e^{2\varepsilon})}{(1 + e^{-s_k})(1 + e^{-s_k - \varepsilon})}. \tag{3}$$

The function is represented by the surface in Fig. 2, where varying values of the error ε and of the input summation s_k are considered. If a step function is taken into account, instead of the sigmoid, the sensitivity of the neuron's output is represented by the surface in Fig. 3.

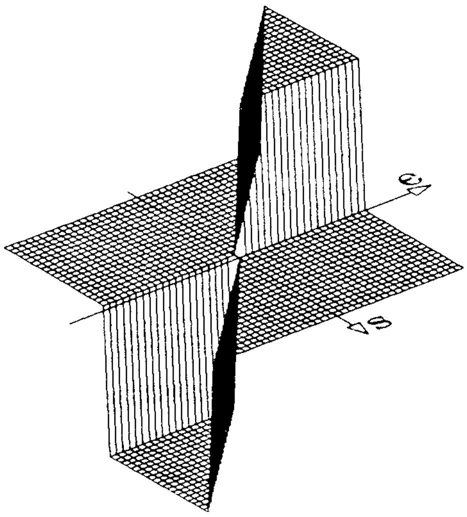

FIG. 3. Local error for the step function.

If the error classes defined above are examined, the following remarks hold:

1. for errors in classes *a* and *b*, the influence on the output of neuron n_k^i depends on the displacement of the operation point on the neuron's evaluation function as a consequence of the error; in fact, due to nonlinearity of such a function, the error can be filtered and an actual error masking is achieved at no cost.

2. errors in class *c* may lead to wider displacement of the neuron's operation point and, thus, are more likely to affect the neuron's output value;

3. errors in class *d* are immediately visible on the neuron's output and no immediate error masking ensues. While this is obviously critical for neurons in the output layer, for the other layers Case 1 will be again considered, with respect to layer $i+1$, where the error affects the inputs of all neurons.

The last case above might lead one to consider, in the following, only the effects of an error on a neuron's output, since ultimately even the other errors, whenever they are not masked, result in this same behavior. At any rate, it is interesting to analyze the extent to which a neuron's behavior is affected by errors on a synapsis (be it the signal transmission, the weight storage, or the weight multiplication which is affected). The analysis is initially limited to the effects of a single error; extension to multiple errors will then be outlined.

In the layered net, assume an error of any of the previous classes affecting neuron n_j^i, i.e., the *j*th neuron of layer *i*. The error affects its output x_j^i; denote the variation of the output value with respect to the expected one as ε. As a consequence, the summation s_k^{i+1} computed by any neuron n_k^{i+1} belonging to layer $i+1$ will be affected by an error. The analysis is first restricted to the effects between the two adjacent layers.

The modified value of s_k^{i+1} is now expressed by

$$\bar{s}_k^{i+1} = \sum_h w_{k,h}^{i+1,i} x_h^i - \theta_k^{i+1} - w_{k,j}^{i+1,i} x_j^i + w_{k,j}^{i+1,i} \bar{x}_j^i, \tag{4}$$

where x_j^i is the expected value of the output of n_j^i in the absence of error, while $\bar{x}_j^i$ is the value of the same signal in the presence of an error (i.e., $\bar{x}_j^i = x_j^i + \varepsilon$). It is then

$$\bar{s}_k^{i+1} = s_k^{i+1} + w_{k,j}^{i+1,i} \varepsilon.$$

The output of any neuron n_k^{i+1} in layer $i+1$ will in turn be affected by an error

$$e_k^{i+1} = \bar{x}_k^{i+1} - x_k^{i+1} = f_k^{i+1}(s_k^{i+1} + w_{k,j}^{i+1,i}\varepsilon) - f_k^{i+1}(s_k^{i+1}). \tag{5}$$

Referring to Cases 1 and 2 above, the instances in which error masking may occur can now be analyzed. Two alternative cases can be considered:

A. the value of the input summation s_k^{i+1} is such that the operation point of n_k^{i+1} falls well within the saturation region of the evaluation function;

B. the value of s_k^{i+1} is such that the operation point falls in the transition region of the sigmoid.

In Case A, denote by μ_k the maximum acceptable absolute value for the output error in the saturation region, i.e., the value such that the neuron's output can be taken as equivalent to the nominal one. Then, denote by $s_k^{i+1,\mu}$ the limit value of s_k^{i+1} such that the neuron's output value will fall within acceptable bounds; obviously, only displacements of the operation point that lead out of the saturation region should be considered. Given the expression of the sigmoid function, it is

$$s_k^{i+1,\mu} = g_k^{i+1}(1-\mu_k),$$

where $g_k^{i+1}(\cdot)$ denotes the inverse function of $f_k^{i+1}(\cdot)$.

Then error masking occurs whenever the operation point remains in the saturation region, i.e.,

$$\bar{s}_k^{i+1} \geqslant s_k^{i+1,\mu}$$

or

$$\bar{s}_k^{i+1} \leqslant -s_k^{i+1,\mu}.$$

Should a step function be used instead of the sigmoid, $s_k^{i+1,\mu}$ obviously becomes null for any value of μ such that $0 \leqslant \mu \leqslant 1$. Whenever these bounds are satisfied, the output error is null.

In Case B, a left and a right bound must be defined for the operation point such that the corresponding output value falls within acceptable, predefined limits. Let two such bounds be

$$s_k^{i+1,\mu,l} = g_k^{i+1}(f_k^{i+1}(s_k^{i+1}) - \mu_k)$$
$$s_k^{i+1,\mu,r} = g_k^{i+1}(f_k^{i+1}(s_k^{i+1}) + \mu_k);$$

then, error masking is granted whenever it is:

$$s_k^{i+1,\mu,l} \leqslant \bar{s}_k^{i+1} \leqslant s_k^{i+1,\mu,r}.$$

It could be deduced that, provided the operation points of all neurons were brought well within the saturation region and the errors affecting the input summation did not lead to displacements larger than the bounds defined above, errors (even multiple ones) could be masked by the intrinsic network's operation (see also [17]). Still, it must be noticed that such policy involves:

1. higher learning time,
2. higher precision in the definition of weights (independent of the implementation technology adopted),
3. higher recall time.

Thus, if reference is made to the classical fault-tolerance approaches, both *time* and *information redundancy* are required.

Moreover, the above conditions for error masking can actually be granted in a statistical way; it is impossible to grant that *any* error will keep the operation point within the predefined bounds.

If the error was *always* null, the neural paradigm should be *completely* intrinsically fault tolerant. By analyzing the above constraints for the envisioned application, the designer could exactly verify if his or her approach is completely fault tolerant or, at least, which is the probability that the possible behavioral errors will be masked. In other words, the designer can evaluate his or her confidence in relying only on the intrinsic ability of the neural computation in order to achieve fault tolerance, independent from both possible learning enhancement and any specific architectural strategy.

3. CONDITIONS FOR FAULT TOLERANCE THROUGH WEIGHT MODIFICATION

To verify from a theoretical point of view whether fault tolerance can be achieved without introducing any structural redundancy in the neural graph (i.e., for example, by enhancing the learning procedure), it is necessary to examine the possibility of obtaining the correct outputs from the fault-stricken network through a suitable redistribution of weights devised to operate on the same set of input classes initially specified for the fault-free network. Permanent, fixed-value errors are assumed; i.e., the error is assumed to remain identical for all input patterns to the stricken neuron or at least for all patterns in each given class. The effects of any error (either within a given neuron or on a synapsis leading to it) are considered to be represented simply by the deviation of the neuron's outputs from the expected value, given the specific set of inputs. This allows us to carry on the following elaboration independent of the particular type of error. Initially, the single-error assumption is adopted: extension to multiple errors will be outlined at the end.

A first, simplified fault-tolerance requirement can be introduced as follows: assuming the presence of a faulty neuron n_j^i in layer i, we want to verify whether its presence can be completely masked at the outputs of all neurons of layer $i+1$ by properly updating the interconnection weights between neurons of layer i and neurons of layer $i+1$. To further simplify operations, a simple form of error confinement is introduced; namely, null weights are set on all the synapses leading from the stricken neuron (fault confinement is an usual prerequisite of fault-tolerance policies). In other words, the interconnection weights between the two adjacent layers must be modified to satisfy the requirement that the error e_k^{i+1} at the output of any neuron n_k^{i+1} of layer $i+1$ be zero for any pattern belonging to the input classes specified for the given network. (It may be noticed here that this policy implicitly does not apply whenever the stricken neuron is in the output layer; this is consistent with the semantics of such a neuron's operation, since the output layer actually provides the coding of the classification performed.)

Assume that $\tilde{s}_k^{i+1}$ is the new value of the input summation after any weight modification $\Delta w_{k,h}^{i+1,i}$ between layer i and layer $i+1$, leading to $\tilde{w}_{k,h}^{i+1,i}$. It is

$$\begin{aligned}\tilde{s}_k^{i+1} &= \sum_h \tilde{w}_{k,h}^{i+1,i} x_h^i - \theta_k^{i+1} \\ &= \sum_h w_{k,h}^{i+1,i} x_h^i - \theta_k^{i+1} + \sum_h \Delta w_{k,h}^{i+1,i} x_h^i \\ &= s_k^{i+1} + \sum_h \Delta w_{k,h}^{i+1,i} x_h^i, \end{aligned} \tag{6}$$

where we consider

$$\Delta w_{k,h}^{i+1,i} = \begin{cases} -w_{k,h}^{i+1,i}, & \text{for} \quad h=j \quad \text{and} \quad \forall k \quad \text{in layer} \quad i+1 \\ \omega_{k,h}^{i+1,i}, & \text{for} \quad h \neq j \quad \text{and} \quad \forall k \quad \text{in layer} \quad i+1 \end{cases} \tag{7}$$

and $\omega_{k,h}^{i+1,i}$ are the unknown modifications of weights required to achieve error masking.

It is immediately stated that error masking through weight updating can certainly be achieved if the values of the input summations $\tilde{s}_k^{i+1}$ obtained by Eq. (6) are identical to the values of the input summations s_k^{i+1} in the absence of errors. In this case, in fact, the outputs of the neurons in layer $i+1$ will be identical to the expected ones independent of the particular position of the operation point or on the shape of the nonlinear evaluation function. Thus, it must be

$$\tilde{s}_k^{i+1} - s_k^{i+1} = \sum_h \Delta w_{k,h}^{i+1,i} x_h^i = 0 \tag{8}$$

which can be rewritten for each neuron in layer $i+1$ by highlighting the unknown quantities, as:

$$\sum_{h \neq j} \omega_{k,h}^{i+1,i} x_h^i = w_{k,j}^{i+1,i} x_j^i. \tag{9}$$

Now let H be the number of neurons in layer i and K the number of neurons in layer $i+1$; Eq. (9) defines a linear system of K equations in the variables $\omega_{k,h}^{i+1,i}$. The number of variables is $(H-1) \times K$, since there are $H \times K$ interconnection weights between layers i and $i+1$, but K weights are fixed by the rule of Eq. (7). The system is linear since the x_h^i do not depend upon the $\omega_{k,h}^{i+1,i}$. Besides, since no modification is envisioned for the synaptic weights in layers preceding layer i, for each given input pattern presented the values of the x_h^i, as well as that of $w_{k,j}^{i+1,i} x_j^i$, may be considered to be constant values.

The linear system defined by Eq. (9) for each given input pattern may be formally compacted by using a matrix description. Let $\mathbf{\Omega} = [\omega_{k,h}^{i+1,i}]$ (with $k \in [1, K]$, $h \in [1, H] - \{j\}$) be the matrix of unknown weight modifications. The row vector $\mathbf{\Upsilon}$ is defined by cascading the rows $\mathbf{\Omega_k}$ of $\mathbf{\Omega}$, i.e., $\mathbf{\Upsilon} = [\mathbf{\Omega_1} \mid \mathbf{\Omega_2} \mid \cdots \mid \mathbf{\Omega_K}]$. The column vector $\mathbf{C}$ is given by the constant vector $\mathbf{C} = [w_{k,j}^{i+1,i} x_j^i]$.

Finally, the coefficient matrix $\mathbf{\Phi}$ is defined as a block-diagonal matrix, where each nonnull block on the main diagonal is equal to the row vector $\mathbf{X_i} = [x_h^i]$ of the output signal generated by the neurons of layer i in the absence of errors. It has K rows and $(H-1) \times K$ columns. It can be proved that the rows of $\mathbf{\Phi}$ are linearly independent, since no linear combination of any set of $(k-1)$ rows can generate the remaining row. This can be easily derived from properties of linear algebra by assuming that at least one x_h^i is not null.

From the previous definitions, the linear system of Eq. (9) is thus given by

$$\mathbf{\Phi} \cdot \mathbf{\Upsilon}^T = \mathbf{C}, \tag{10}$$

where $\mathbf{\Upsilon}^T$ is the transpose of $\mathbf{\Upsilon}$.

Equation (10) has a unique solution for a given input pattern iff $\mathbf{\Phi}$ is a square matrix and may be inverted, i.e., $K = (H-1)\,K$ (namely, $H=2$, $K>0$) and $\det(\mathbf{\Phi}) \neq 0$. In such a case, the weight modifications that confine the error between the two adjacent layers are given by $\mathbf{\Upsilon}^T = \mathbf{\Phi}^{-1}\mathbf{C}$. Note that, when $H=2$, matrix $\mathbf{\Phi}$ becomes a diagonal matrix and, thus, $\det(\mathbf{\Phi}) \neq 0$ iff $x_h^i \neq 0$, with $h \neq j$ (i.e., the output signal of the error-free neuron is not null in the absence of errors). The probability p that a solution exists is given by $p = p(x_h^i \neq 0 \mid h \neq j)$.

If $\mathbf{\Phi}$ is a rectangular matrix, the number of columns is larger than that of rows ($H>2$ and $K>0$), and an infinite number of solutions occurs. In fact, if a square submatrix $\mathbf{\Psi}$ (having rank K) of $\mathbf{\Phi}$ can be found to be invertable, an infinity of solutions of order $(H-2)\,K$ exists. This means that, for the given input pattern, the value of $(H-2)\,K$ weights can be arbitrarily chosen. Due to the characteristics of matrix $\mathbf{\Phi}$, it is possible to show that the probability p that a solution exists is the probability that at least one x_h^i (with $h \neq j$) is not null; i.e., $p = 1 - \Pi_{h \neq j} p(s_h^i = 0)$, by assuming that x_h^i are mutually independent.

Extension of Eq. (10) to the whole set of P input pattern classes may be achieved by considering a set of Eq. (10) for each class p (obviously, the values in the constant vector will be different for each pattern as will the signals output from layer i). This leads to

$$\tilde{\mathbf{\Phi}} \cdot \mathbf{\Upsilon}^T = \tilde{\mathbf{C}}, \tag{11}$$

where $\tilde{\mathbf{\Phi}}^T = [\mathbf{\Phi_1}^T \mid \mathbf{\Phi_2}^T \mid \cdots \mid \mathbf{\Phi_P}^T]$, $\tilde{\mathbf{C}}^T = [\mathbf{C_1}^T \mid \mathbf{C_2}^T \mid \cdots \mid \mathbf{C_P}^T]$, and $\mathbf{\Phi_p}$ and $\mathbf{C_p}$ are matrices $\mathbf{\Phi}$ and $\mathbf{C}$ (defined above) for class p.

Equation (11) defines a linear system of PK equations in $(H-1)\,K$ variables: its solution (if it exists) gives the weight modifications which guarantee *complete* tolerance to the injected error for all input classes. If no solution exists, no learning procedure—even if leading to an enhanced masking ability—will be able to *completely* grant fault tolerance by relying on the intrinsic masking ability only; as a consequence, the designer should carefully consider the opportunity of adopting architectural strategies for fault tolerance with respect to the specific requirements of the envisioned application. In this system, some rows may be a linear combination of other rows, so that the actual number of equations that should be simultaneously solved will be lower than PK. It can be proved that, if two rows are found that are mutually linearly dependent, linear dependency can be found between $2P$ rows. Should the determinant of the system be null, a possible alternative in order to reach a solution could be to move a small distance to the center of each class, thus slightly modifying the parameters of the system without affecting in a relevant way the classification capacity.

Extension to the instance of multiple errors is straightforward; if the errors are all located in one layer, the condition stated by Eq. (7) is applied for all stricken neurons (this obviously reduces the number of variables). Otherwise, the set of equations will have to be determined independently for each pair of layers between which errors are injected, keeping in mind that stricken neurons cannot actively participate in the weight updating operations.

If the previous constraints upon $\mathbf{\Phi}$ are not satisfied, no weight modification can be found for the given input patterns to grant error masking already at the inputs of the evaluation functions in layer $i+1$. Actually, this does not immediately exclude the possibility of error masking through a more extended propagation of weight updating through *all* layers from $i+1$ to the output one. An attempt at obtaining such a satisfactory modified weight distribution should be made; still, it is quite obvious that an exact solution would be very hard to determine, due to the presence of nonlinear functions. An approximate solution will be therefore considered by first introducing constraints that will make the problem manageable.

The analysis is restricted to the instance of errors whose effects are sufficiently limited to keep the operation point (for all neurons in the layers following the stricken neuron) in a small interval around its nominal position on the nonlinear evaluation function. As already stated, continuous, derivable functions are envisioned; it is then acceptable to substitute, within the limited interval now defined, the nonlinear function with its linearization, i.e., the tangent in the nominal operation point. Assuming again an error ε in the output x_j^i of neuron n_j^i, the error at the output of any neuron n_k^{i+1} is now expressed as:

$$\begin{aligned} e_k^{i+1} &= \bar{x}_k^{i+1} - x_k^{i+1} = f_k^{i+1}(s_k^{i+1} + w_{j,k}^{i,i+1}\varepsilon) - f_k^{i+1}(s_k^{i+1}) \\ &\simeq \left.\frac{df_k^{i+1}}{d\lambda}\right|_{\lambda = s_k^{i+1}} w_{j,k}^{i,i+1}\varepsilon. \end{aligned} \tag{12}$$

The above holds for the individual pattern corresponding to the operation point taken into account; thus, Eq. (12) must be repeated for all pattern classes. Unless layer $i+1$ is the output one, Eq. (12) must be extended for all subsequent layers and for each $m > i+1$

$$\begin{aligned} e_k^m &= \bar{x}_k^m - x_k^m = f_k^m\left(s_k^m + \sum_h w_{h,k}^{m-1,m} e_h^{m-1}\right) - f_k^m(s_k^m) \\ &\simeq \left.\frac{df_k^m}{d\lambda}\right|_{\lambda = s_k^m} \sum_h w_{h,k}^{m-1,m} e_h^{m-1} \end{aligned} \tag{13}$$

is obtained where $\bar{x}_k^m$ denote the erroneous values of the outputs produced by each neuron in layer m. A more compact expression of (13), making use of vector notation, is

$$\mathbf{E_m} \simeq \mathbf{F'_m}|_\mathbf{S} \mathbf{W_{m,\,m-1}} \mathbf{E_{m-1}}, \tag{14}$$

where $\mathbf{E_m}$ is the vector of output errors at layer m, $\mathbf{S}$ is the vector of the summation inputs, $\mathbf{F}'$ is the vector of derivatives of $\mathbf{F}$ in $\mathbf{S}$, and $\mathbf{W_{m-1,\,m}}$ is the weight matrix between the adjacent layers. It should be noted that Eq. (13) is a recurrent equation whose solution may be achieved by iterative substitutions and involves a relevant computational complexity.

Having thus determined the error propagation through the system, it is necessary to attempt variations of weights throughout all layers (backward from the output layer up to, and including, layer $i+1$) to set an upper bound on the final error produced by the net. To this purpose, two alternative aims are considered: either setting such error to 0 or else minimizing the mean square of the error itself.

Variation of the weights is achieved by considering each weight $w_{k,h}^{i+1,\,i}$ to be modified by $\Delta w_{k,h}^{i+1,\,i}$; thus, the input summation to the evaluation function for each neuron in layer $i+1$ is:

$$\tilde{s}_k^{i+1} = s_k^{i+1} + w_{k,j}^{i+1,\,i}\varepsilon + \sum_h \Delta w_{k,h}^{i+1,\,i} x_h^i + \Delta w_{k,j}^{i+1,\,i}\varepsilon. \tag{15}$$

As a consequence, the variation of each neuron's output value is

$$\Delta x_k^{i+1} = \tilde{x}_k^{i+1} - x_k^{i+1} = f_k^{i+1}(\tilde{s}_k^{i+1}) - f_k^{i+1}(s_k^{i+1}) \tag{16}$$

which (assuming small errors) can be approximated as:

$$\Delta x_k^{i+1} \simeq \left.\frac{df_k^{i+1}}{d\lambda}\right|_{\lambda = s_k^{i+1}} \left[(w_{k,j}^{i+1,\,i} + \Delta w_{k,j}^{i+1,\,i})\,\varepsilon + \sum_h \Delta w_{k,h}^{i+1,\,i} x_h^i \right]. \tag{17}$$

Equation (17) must now be extended to any layer subsequent to $i+1$; in this phase, second-order differences would appear that are here assumed negligible by comparison with the other terms. Thus, finally, the variation of each neuron's output in any layer m, $m > i+1$, is evaluated as:

$$\Delta x_k^m \simeq \left.\frac{df_k^m}{d\lambda}\right|_{\lambda = s_k^m} \left[(w_{k,h}^{m,\,m-1} + \Delta w_{k,h}^{m,\,m-1})\,\varepsilon + \sum_h \Delta w_{k,h}^{m,\,m-1} x_h^{m-1} \right]. \tag{18}$$

Using vector notation, Eq. (18) can be written as

$$\mathbf{\Delta X_m} \simeq \mathbf{F'_m}|_\mathbf{S} (\mathbf{W_{m,\,m-1}}\,\mathbf{\Delta X_{m-1}} + \mathbf{\Delta W_{m,\,m-1}} \mathbf{X_{m-1}}), \tag{19}$$

where $\mathbf{\Delta X_m}$ is the vector of output variations due to the injected errors and to the weight modifications, $\mathbf{\Delta W_{m,\,m-1}}$ is the matrix of weight modifications between the two adjacent layers, and $\mathbf{X_{m-1}}$ is the vector of outputs generated by neurons belonging to layer $m-1$. Again, Eq. (18) concerns just one pattern and it should be extended—as in the previous case—to all pattern classes taken into account.

By applying Definition (7), the variables ω of the unknown modifications of weights are introduced.

By iterative substitution in the recurrent equation thus obtained, the following system of linear equations can be reached.

$$\mathbf{\Delta X_M} = \mathbf{C}\,\mathbf{\Delta X_j} + \sum_{m=i+1}^{M} \mathbf{\Delta W_{m,\,m-1}}\,\mathbf{D_m}, \tag{20}$$

where $\mathbf{C}$ and $\mathbf{D}$ are constant vectors depending on the input classes. The injected error is completely masked at the final outputs of the neural net if $\mathbf{\Delta X_M} = 0$. This leads us to rewrite Eq. (20) to achieve a linear system similar to Eq. (11). Solutions, constraints, and remarks on the use of enhancing fault tolerance are thus similar to those discussed above for such a system. Note that if no solution exists, classical dynamic programming techniques (e.g., see [24]) can be applied to identify a set of values for the variables ω that minimize the objective function $\phi = \min(\|\mathbf{\Delta X_M}\|)$. This will be the best possible masking achievable by weight redistribution and, in turn, by a learning procedure aiming to enhance the intrinsic masking ability of the neural computation; however, *complete* masking is not provided and the designer should consider the necessity of architectural supports to fault tolerance.

Should the linearization adopted until now (i.e., the small error assumption leading to Eq. (17) and the subsequent analysis) be incompatible with the actual variation of operation points (as verified a posteriori on the network with the modified weight values), Eq. (20) should have to be treated not as a linear recurrent equation but as a nonlinear equation—with the obvious increase in difficulty. Unfortunately, this case often occurs in digital implementations, making the theoretical feasibility analysis for enhancing the intrinsic masking ability by weight adjustment impractical.

Extension to the case of multiple errors is simple in the assumption that their global effect is still limited enough to allow linearization around the nominal operation point for each fault-free neuron. In this case, effects of the individual errors are simply superimposed to reach the expression of the complete case.

4. SIMULATIONS AND EXPERIMENTAL RESULTS

The analytical treatment presented in the previous sections allows us to understand the influence and the relevance of behavioral errors on the neural computation as well as the mathematical conditions that have to be satisfied in order to grant fault tolerance without introducing structural redundancy in the architecture implementing the neural network. In other words, we evaluated the conditions for which fault tolerance can be achieved by using a suitable learning technique (e.g., [19]), since the weight adjustment can be viewed as a learning phase, during which the nominal training pattern classes are used to obtain correct responses even in the presence of errors due to faults within the network. While this kind of learning involves a time overhead related to the additional training patterns due to the presence of errors, it cannot be actually considered a time-redundancy policy, since the time overhead is introduced only during weight configuration and does not influence the subsequent recall phase.

The conclusions reached in Section 3, even though interesting in that they allow us to state that fault tolerance *can* be reached under very specific constraints but it is not a general property of the neural paradigm, suggest that use of mathematical equations to evaluate both these conditions and the modified weights would be extremely time-consuming.

Therefore, a simulation-based analysis is performed in this section to show a more practical way that can be adopted by the designer in order to verify the fault-tolerance characteristics of his or her network, the possibility of intrinsic error masking in the presence of faults, and the possibility of enhancing such an ability by adopting a suitable weight adjustment technique (either a priori during learning [19] or after fault occurrence by relearning the desired behavior in the presence of a faulty architecture). It is worth noting that simulation allows also for a quantitative evaluation of the masking ability. Simulations were carried out over several networks, all trained over the same set of input patterns and therefore characterized by an identical output layer, but otherwise differing for the number of neurons in the hidden layer(s), the number of layers, and the distribution of neurons among the various hidden layers. This allows us to evaluate the influence of such structural characteristics on the network's behavior in the presence of errors.

The network performances are defined with reference to the capacity of effecting a classification of randomly generated input patterns with respect to a given set of classes. In turn, to grant the greatest generality to the experiment, each class is characterized by a central representative and by a set of related patterns generated by random (white) noise within a given (even reasonably large) range. An example of the tessellation of the space of classified patterns is given in Fig. 4 for a set of six classes (the one used for the presented experiments).

The multilayered nets devised to classify such patterns all have at least one—and possibly more—hidden layer. Obviously, all nets have as many neurons in the output layer as the number of classes. Among the different nets, a minimum-complexity network (i.e., a network capable of granting correct classification over the given input set but such that nets with less neurons could not achieve such a classification) is created in an experimental way by applying any neural structure minimization technique [25]. The evaluation function adopted is a sigmoid for all but the output layers; a step function is adopted for the output layer to achieve a binary output value. All other output signals have continuous values ranging between -1 and $+1$.

A random distribution of weight values, ranging between -1 and $+1$, is initially provided over the untrained network; the first learning (performed over the fault-free network, prior to error injection) leads to the achievement of the proper weight distribution related to the given pattern classes. The final values of weights may be arbitrary real numbers. The learning algorithm is a classical back-propagation one [23]: different values have been used for the algorithm parameters to check the sensitivity of the evaluations to the characteristics of the algorithm.

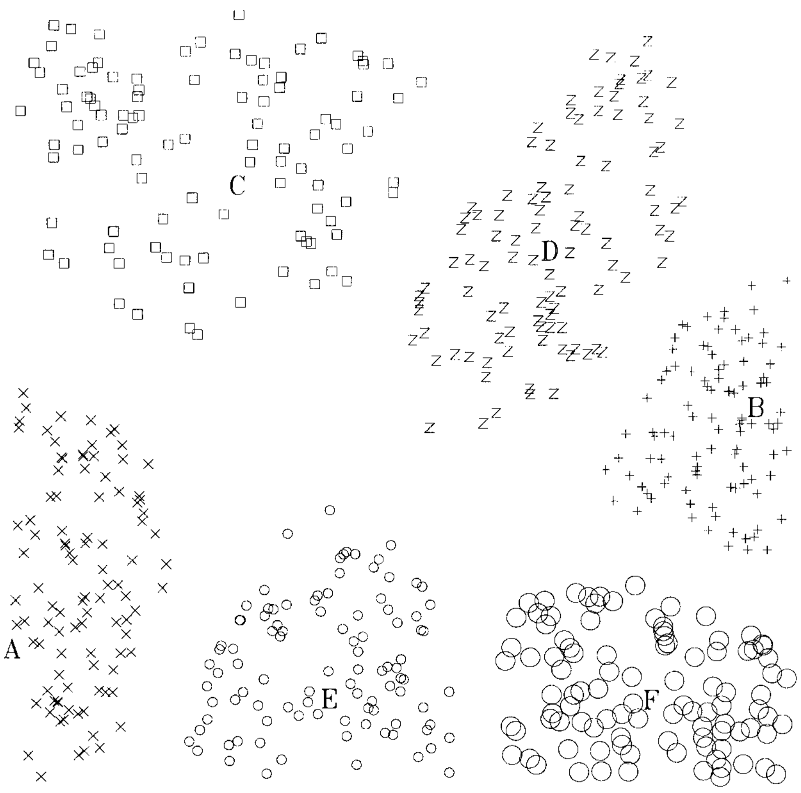

FIG. 4. Classification space.

Two separate error instances were considered that effectively can be seen to cover the error classes introduced in Section 2. The first instance concerns errors affecting individual input signals, individual synaptic weights, or weight-signal multipliers. Both single and multiple errors will be taken into account (note that this could include also some errors of the summation unit). The second instance concerns errors affecting the evaluation function (again, relevant errors affecting the summation can be experimentally seen as creating similar effects).

Two alternative policies have been adopted for defining the value of the errors to be injected. The first refers to an additive error summing to the correct value found on a synapsis or on the output of a given neuron; the second one consists in a fixed value stuck-at a synapsis or on the output of a neuron, whatever the ideal value of such output. These two instances, while coherent with the error model defined in the second section, are compatible with fault instances in silicon implementations as well. Simulations have given practically identical results for both error types; as a consequence, only the set of results derived from the first alternative will be presented here.

Refer first to a simulation of the net's behavior in the presence of a single error. Nets with three layers (Case a) and with five layers (Case b) are analyzed. The number of pattern classes is six for all instances. For Case a, the behavior of a large number of nets is examined with varying numbers of neurons (starting from the lowest one, as defined above, to a number approximately double that) and with different distributions of such neurons among the three layers, so as to account for different amounts of synaptic weight memory. Different structures (both with respect to the number and to the distributions of neurons) were examined also for the five-layer nets, although (due to the complexity of the experiment) the analysis was not as exhaustive as in the three-layer case.

A further element of discussion is the position of the error-stricken neuron within the net; thus, for example, when a three-layer net is considered, it can be seen that while the position of the fault within a layer is not relevant (as it is self-evident), the particular layer in which the fault is positioned is relevant to the value of the final classification error.

Effects of synaptic errors are examined initially; intuitively (and also from the mathematical analysis performed above), such errors are expected to have a relatively small influence on the network's behavior, but are possibly nonnull. In Fig. 5, the effects of a single synaptic error are represented for a number of three-layered nets, starting from the minimum-complexity one and going toward increasing numbers of neurons in the input and hidden layers; the two axes in the horizontal plane show the number of neurons in the input (layer 1) and in the hidden (layer 2) layers, respectively, while the quote is the observed maximum error. Figure 5a gives the residue classification error in the absence of faults, assuming that learning is continued until no improvement is achieved and trying to reach total classification capacity; Figure 5b gives the classification error due to a fault in a single synapsis randomly located. Identical examples have been taken into

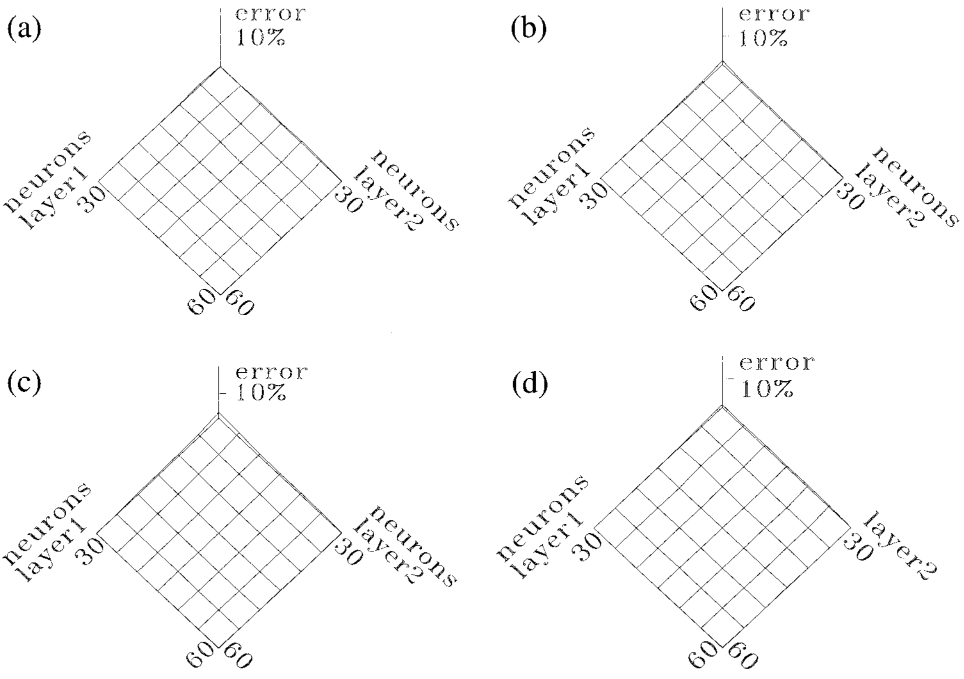

FIG. 5. Error due to faulty synapsis: after initial learning (a), with one fault after fault injection (b), with multiple faults after fault injection (c), and with one fault after repeated learning (d).

account for multiple synaptic errors, still keeping the multiplicity low with respect to the network's connectivity; effects are given in Fig. 5c. The simulation results confirm the initial assumption of a very reduced influence of such errors on the neural computation, even if it is not null.

Then, the effects of the second error instance, namely that by which a whole neuron can be considered as error-stricken giving an unexpected value on its output, are analyzed. In Fig. 6a, the percentage error of classification is given for a three-layer net with a varying number of neurons in the input and in the hidden layer when the fault is positioned in the input layer. The high error ratio when the number of neurons in the first and second layers is small is to be expected, since obviously the corresponding network is close to its minimum complexity and no intrinsic redundancy is present. The apparently surprising increase in error for *high* numbers of neurons can be justified by the sensitivity of stable configurations to *highly redundant* net structures. In any case, the values obtained for errors are quite high, surprisingly so in view of the general assumption of intrinsic fault tolerance for neural nets.

Similar simulations have been performed also for networks with *unperfected* learning in the absence of faults. In this case, the *initial* classification error is not null since learning has been stopped as soon as the classification error dropped below a predefined bound (namely, 4% in our experiments). As it was to be expected, the effect of the fault is magnified in these conditions.

In Fig. 7 the effect of a faulty neuron in the hidden layer is represented, again with respect to different network structures, while in Fig. 8 the fault has been positioned in the output layer (these two instances are computed for an initial 100% classification capacity).

All the above simulations lead to the deduction that neural networks may be able to mask some classes of errors (at least up to a given degree) due to the intrinsic characteristics of the neural computation. But this ability does not hold for *all* kinds of error and for *all* error magnitudes. As a consequence, neural networks cannot be considered *completely* intrinsically fault tolerant at the behavioral level.

The effect of repeated learning as a weight adjustment technique is now examined to evaluate the limits of techniques enhancing the intrinsic masking ability with respect to different error classes. Simulations have been performed for all the

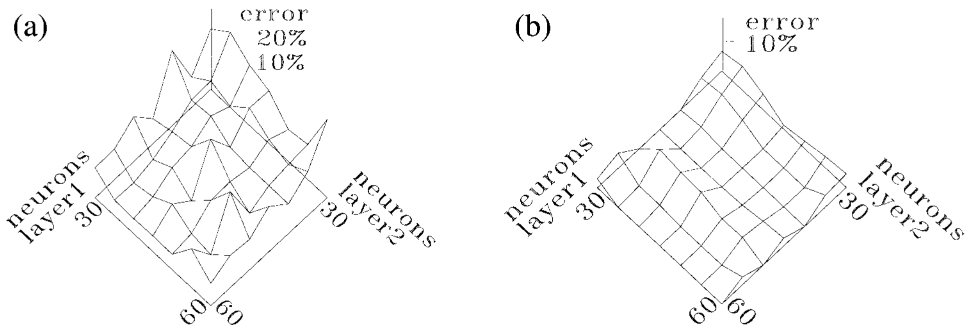

FIG. 6. Error due to a faulty neuron in the input layer: after fault injection (a) and after repeated learning (b).

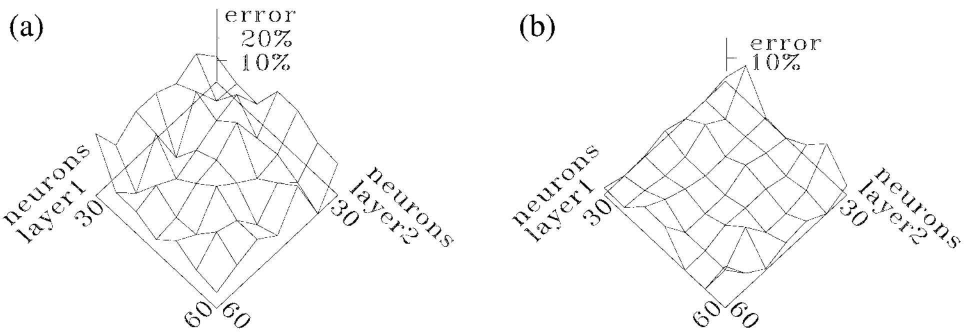

FIG. 7. Error due to a faulty neuron in the hidden layer: after fault injection (a) and after repeated learning (b).

previous examples: the corresponding surfaces are represented in Figs 5d, 6b, 7b, and 8b. With respect to the synaptic errors, given their modest initial relevance a repeated learning phase grants almost complete error masking, as could expected, but not exactly complete, as desired. Concerning the neuron errors, it should be noted that, even after repeated learning, the residue error with partial learning (i.e., when the learning algorithm is stopped as soon as the actual computation error is lower than a given upper bound and 100% initial classification had not been reached) is much higher than in the case of complete learning (i.e., when the learning algorithm runs until the residue error is steady). This can be interpreted as a lower capacity of error recovery through repeated learning when the initial learning was not definitely perfected. Moreover, as could be intuitively inferred, a faulty neuron in the output layer is much more critical (even in the presence of repeated learning) than one in the other layers, since weight redistribution has no actual meaning in this case.

Simulations of the same type were then repeated by inserting two error-stricken neurons; the corresponding classification errors are given in Figs. 9a and 10a for different locations of the stricken neurons (clearly, it is useless to repeat the surfaces corresponding to an absence of faults) while the situation after repeated learning is represented in Figs. 9b and 10b.

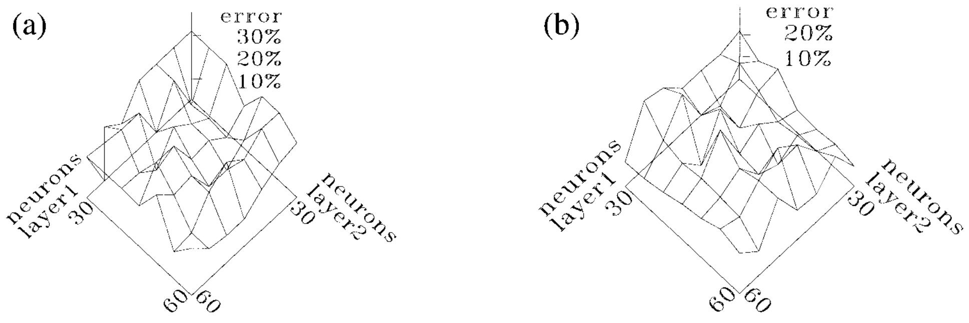

FIG. 8. Error due to a faulty neuron in the output layer after fault injection (a) and after repeated learning (b).

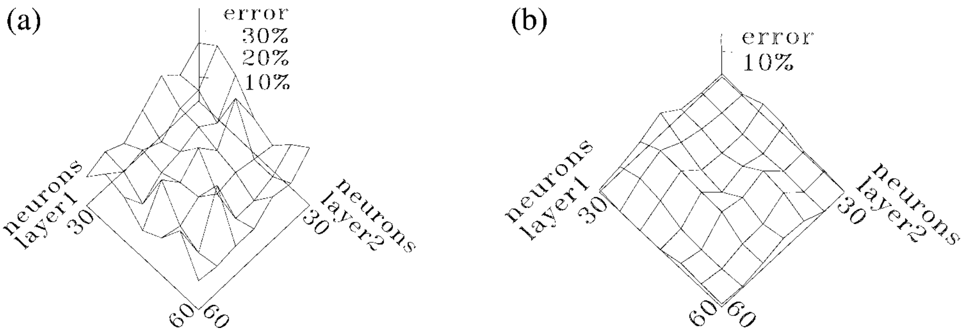

FIG. 9. Error due to two faulty neurons in adjacent layers (one in the input layer, one in the output layer): after fault injection (a) and after repeated learning (b).

Finally, errors and classification capacity after repetition of learning have been derived for five-layer networks, always given the same classes of input patterns. Here, synaptic errors have not been taken into account, given the previous results: in fact, this final study was performed with the main scope of verifying whether the multiplicity of layers affected the behavior in the presence of errors, and therefore only errors producing relevant effects had an actual interest. Simulations carried out allow us to draw meaningful conclusions, namely:

a. results are similar to those obtained for three-layered nets;

b. if the five-layered net has a comparable number of neurons as the functional equivalent three-layered one, the input layer will have a lower number of neurons and therefore it will be a rougher tessellation of the pattern class space. As, it is to be expected, therefore, the effect of a fault in such a layer is more relevant than in the three-layer case. Moreover, the effect of the fault is further magnified by its propagation through a higher number of layers;

c. increasing numbers of neurons in the input layer, with the given classification requirements, allow us to reach better fault-tolerance; this was also to be expected, since the identical initial tessellation is implemented with a larger global redundancy of the network.

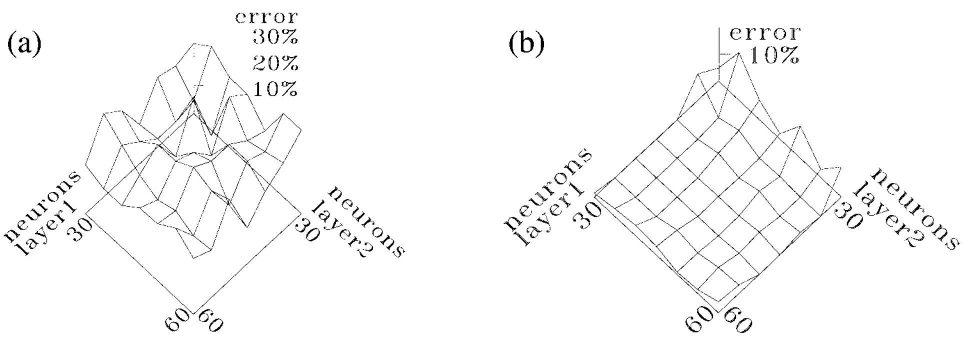

FIG. 10. Error due to two faults in the input layer: after fault injection (a) and after repeated learning (b).

The above analysis of weight redistribution through repeated learning proved that learning (even in the enhanced versions) is not always able to guarantee a *complete* masking ability of the errors induced by any class of faults and by any error magnitude. As a consequence, still neural networks cannot be claimed in general to be able to recover from errors induced by faults by exploiting the intrinsic characteristics of the neural computation and its configuration through learning in the presence of faults. This can be true, but under the strict conditions discussed in Section 3. In other words, the neural network holds the error masking ability only if the faults occurring in the implementing architecture induce errors that satisfy the above constraints. Unfortunately, this is not a very frequent case, especially in the dedicated digital realizations.

Obviously, acceptability of the amount of errors due to faults is strictly related to the specific application and to the relative value with respect to the learning error (i.e., the behavioral error measured in validation at the end of learning as a result of imprecise generalization ability). If the error is sufficiently small for the application, the designer can choose to rely only on the intrinsic masking ability of the neural computation; otherwise, a suitable hardware support must be introduced at the architectural level to guarantee error detection and, possibly, correction. To verify the actual incidence of the faults on the nominal computation and the acceptability of induced errors, the designer should perform simulations similar to the ones presented here by taking into account the probability distributions of the error classes and the magnitude due to the actual probability distributions of faults in the implementing architecture.

5. MAPPING OF FAULTS ONTO BEHAVIORAL ERRORS IN VLSI IMPLEMENTATIONS

To verify the actual influence of faults onto the neural computation by using either the theoretical analysis or the experimental simulation, the designer must map first the faults on the behavioral errors in order to understand their effects on the computation. Then, he or she must transform the fault probability distribution (which specifies the space distribution in the circuit implementing the neural network for each class of faults) into the corresponding error probability distribution at the behavioral level in order to evaluate the relevance of each class of error in terms of frequency of occurrence and, as a consequence, to decide which classes are more dangerous and need specific architectural supports to be dealt with.

A number of silicon implementations of neural networks have been presented in the literature and various devices are even commercially available. In this section, some of these implementations are analyzed from the point of view of the fault-tolerance capabilities treated above so as to show, in particular, how the mapping of hardware faults onto behavioral errors must be performed. The critical parts of each of these implementations are identified and discussed.

The alternative approaches proposed in the literature (e.g., [26–34]) differ not only in the implementation technology adopted (analog vs digital solutions) but also with respect to the philosophy underlying the realization. While some architectures simulate the nets' functions (so that no one-to-one correspondence between a single neuron's functions and a corresponding component's functions can be created) others emulate the individual neurons; again, while in some solutions the complex neural connectivity is mapped onto an identical physical connectivity [26] in other solutions the logical data transfers are time-multiplexed onto a simpler physical interconnection structure. Of the many silicon architectures, four are selected as representative of some main approach: the basic characteristics shared by most analog implementations and three digital architectures, each related to some typical implementation problems.

Analog architectures in general do not involve any form of time-multiplexing over the various components that implement the abstract operators present in the neural network [27]. Physical faults affecting the logical components of such structures directly map onto the behavioral errors listed above, in a one-to-one correspondence. Failures of individual synapses or individual multipliers can well be modeled as discussed in Section 2, and mutual independence between faulty devices (if multiple errors are to be taken into account) can safely be assumed. Usually, as seen from simulation analysis, these faults are masked by intrinsic capabilities of the neural computation, when enough information redundancy has been provided through outputs' saturation: a fault will not result in global failure, but it will simply degrade the system's operation, often leaving the possibility of recovery intact. Faults in electrical components related to system-level activities (e.g., the power supply, the weight-value refresh circuit, and the shifters used to extract the system's outputs) are more critical since they affect in a fatal way the operation of the whole system; it may be observed anyway that such global system elements are found in any type of implementation and that robust design techniques can be adopted for them.

The above considerations can be applied also to such digital architectures where the intrinsic network parallelism is mapped onto the architecture without recurring to time multiplexing; such is the case of the solution presented in [26], where large pipelined binary trees are used to create the cells.

If more general digital solutions are considered, high system survival may often be achieved at the architectural level by using reconfiguration policies [35]; unfortunately, possible time multiplexing of some components to overcome the connectivity requirements can transform a single physical fault into a multiple behavioral error, thus affecting even in a relevant way the system's operation. Obviously, in all digital architectures, faults in power supply and synchronization signals definitely affect the neural computation by generating a system failure.

The first class of digital architectures examined here is based upon systolic array structures which simulate the computation of ANNs. Examples of this approach are given in [28, 34]. For this type of architecture, the case of multilayered back-propagation networks is considered (see Fig. 11); the individual processing node implements the functions of an associated neuron, and the synaptic weights associated with the incoming synapses are stored within the corresponding nodes in a recirculating memory. The neurons' outputs are fed from one layer into the neurons of the subsequent layer and are propagated along the latter layer until all weighted sums have been computed: only when the whole memory has recirculated will the nonlinear function finally be evaluated. Reduced complexity of interconnections

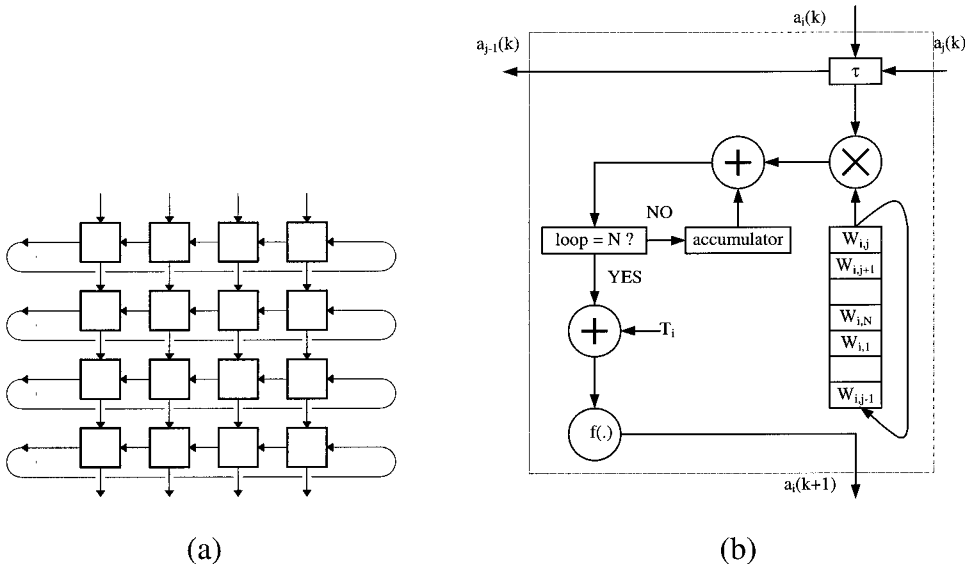

FIG. 11. Kung's architecture.

between neurons is balanced by intrinsic fragility from the point of view of fault tolerance. All internal node faults (e.g., a fault in one word of the weight memory or a multiplier fault) affecting the neural computation result in a global neuron fault. If the node is able to exclude itself from the computation whenever a fault is detected (by testing or online detection techniques [5, 7–12]), still granting correct operation of the bypass switches that propagate the firing signals, a node's faults can be simply mapped onto a stuck-at zero single-neuron error of the behavioral error model; otherwise, much more complex stuck-at models must be introduced on the synapses. Faults in interconnection links between adjacent layers, if conventional fail-safe design techniques have been adopted, will appear as a stuck-at on the output of the origin node and as a consequence they will be collapsed into the single-neuron error class. Catastrophic consequences occur if a fault affects a link that propagates input signals throughout a whole layer: *all* nodes in that layer will receive a whole sequence of incorrect activation signals, and a massive failure will ultimately result.

A second class of digital implementations considers simulation of the neural network by distributing the computation in the whole array instead of associating neurons to array nodes [29]. No mapping (possibly through time multiplexing) of synapses onto interconnection links can be identified: a row of the rectangular array is dedicated to *simulation* of an individual neuron (see Fig. 12); the array computes the product between the weight matrix and the network state, while the activation signals are finally evaluated by the output column. While a single fault in the output column maps onto the error of a full neuron, a fault in any cell of the rectangular array corresponds to an error in the summation input to the evaluation of the activation function. It should be noted that the relevance of such an error depends not only on the proportional relevance of the value computed inside the node with respect to the total summation, but also (and this is more critical with respect to fault tolerance) with the position of the node in the row. Still, it can be

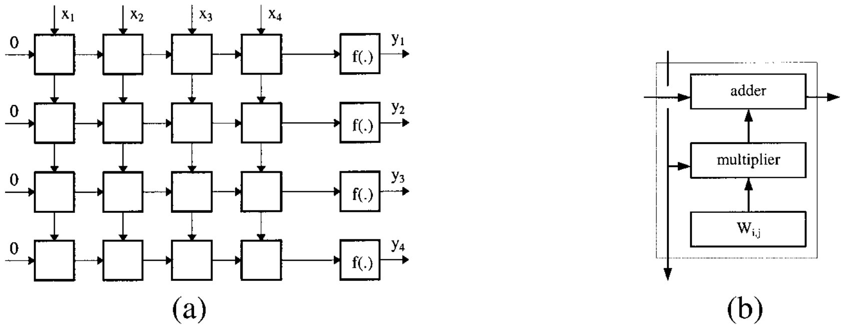

FIG. 12. The neural architecture by Lehmann and Blayo.

added that multiple faults within a row map onto a single behavioral error; moreover, even a fault in an interconnection link maps simply onto the error of one neuron. If suitable fault-confinement policies are adopted for the circuit implementations, many faults can be reduced to pruning a number of synapses leading to a specific neuron in the net. This architecture is thus highly robust since there is no critical component whose failure is catastrophic due to the distribution of computation and to the locality of communications.

The third digital class is based upon array architectures with switched-bus structures. While it is quite obvious that in all such instances the buses become the hard-core of the architecture as far as reliability is concerned, fault-tolerance policies devised for this class of arrays may be adopted (although it might be recalled that faults arising in the interconnection network are usually excluded from reconfiguration policies) as much less probable than faults in processing nodes [35]). This philosophy was adopted, e.g., in [36], where a first mapping of the neural network onto an ideal strip architecture (subsequently folded along the dimensions of a rectangular array) allows one to overcome a comprehensive set of physical faults by exploiting the characteristics of layout and mapping at the same time. In that case, as in other similar ones, mapping of physical onto behavioral faults is not relevant, since the effects of faults are overcome at the architectural level independent of the semantics associated with the faulty devices. A different approach has been adopted in the neurocell array [30], a semi-custom approach onto which the given application is mapped at production time by customizing interconnection routing, contents of the weight memories, and control signals within the individual nodes (see Fig. 13). The array consists of pseudo-neurons, each provided with a local weight memory and with a processing unit which is capable of either evaluating the summation of weighted inputs (accumulated, if necessary, to a similar summation provided by a previous pseudo-neuron) and forwarding this value to the subsequent pseudo-neuron or performing besides this linear operation also the evaluation of the nonlinear function. Each neurocell can perform as a complete neuron or as a pseudo-neuron in any position of a chain making up a complete (larger) neuron. Input signals to the various pseudo-neurons are fed through a stack of buses, through suitable delays, and output signals are fed by the final cell of each neuron onto one of a stack of output buses. A fault in one word of a weight memory maps onto a percentage error of a single synaptic weight; ifs effects can be

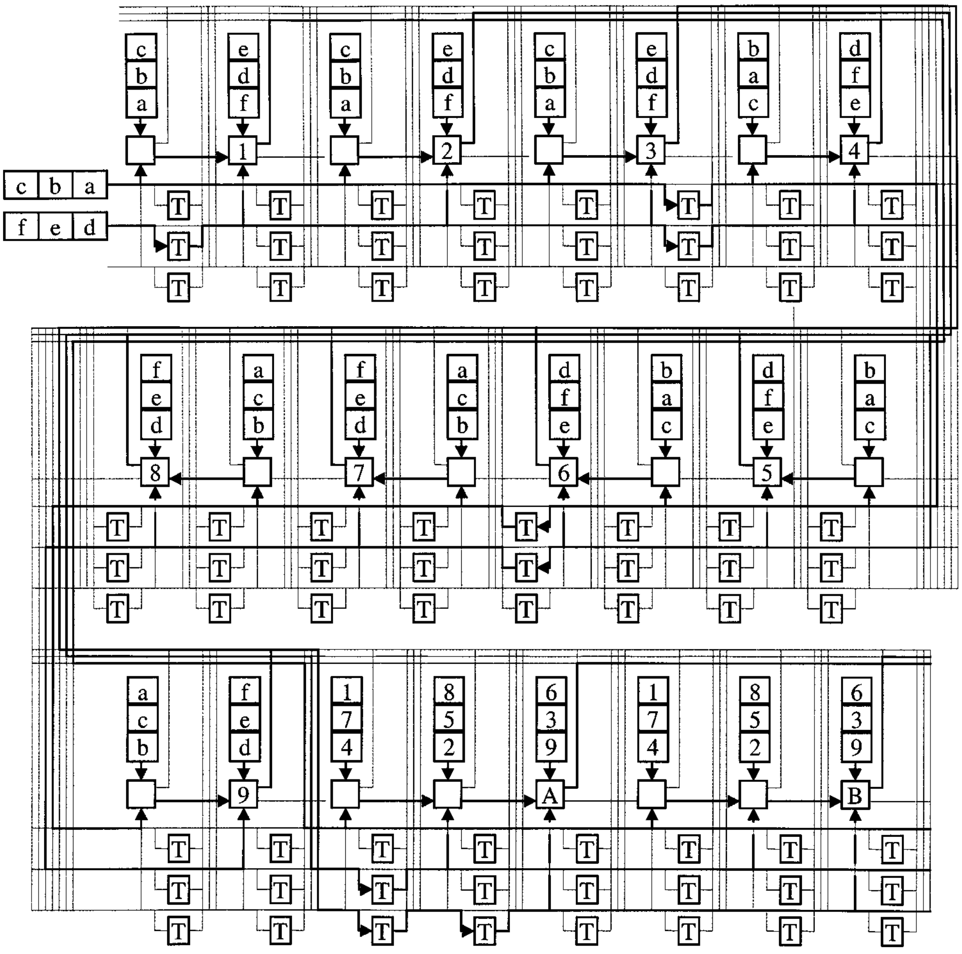

FIG. 13. The neurocell architecture.

simulated and—if the defect is present at production time—a modified weight distribution can be a priori evaluated. A fault of the processing unit or a global fault of its weight memory corresponds to an error in the summation input as far as the set of synaptic weights stored in that neurocell are concerned; if isolation of the faulty neurocell from the array buses is granted, then a pruned neural net is obtained in which one neuron of a given layer is connected only to a subset of the neurons in the previous layer. A fault on an input bus corresponds to pruning all synapses coming from the same subset of neurons in the previous layer leading to the subset of neurons fed by the bus itself. While a relevant error, it does not have the global effects seen previously in the instance of the systolic-array mapping; in fact, while time-multiplexing is present here also, it involves for each multiplexed bus only a subset of both origin and destination nodes. Unless both origin and destination nodes consist of single neurocells, a measure of computational capacity will survive and (depending on the intrinsic redundancy of the network) an attempt at modified weight evaluation can be made. Faults of an output bus correspond to the failure of all neurons that feed their output signals onto this bus; as in the previous case, this is again a multiple but not necessarily a fatal failure. Further robustness and system survival can be achieved by adopting reconfiguration policies at the array level.

6. DESIGN GUIDELINES AND CONCLUSIONS

Theoretical considerations and simulation results on the effect of errors in multilayered feed-forward neural nets allowed us to reach a number of conclusions, sometimes contrasting with intuitive evaluations currently proposed. Errors affecting the synapses have been very limited consequences—even multiple errors may imply no, or very little, modification of the network's behavior. On the contrary, errors affecting the global behavior of even one single neuron easily lead to relevant classification errors during the network's operation, although the network may be structurally redundant with respect to the required nominal operations.

Mathematical conditions under which the distribution of weights can be modified so as to mask the error have been derived. This weight redistribution can actually be achieved, rather than by the analytical solution of a set of complex equations, by a repeated learning phase or by applying a priori a learning technique oriented to enhance intrinsic fault tolerance. The outcome of such a policy has been verified by a number of simulation experiments. On the other hand, it is worth noting that if the mathematical conditions mentioned above are not satisfied, there is no weight redistribution (and, in turn, learning technique) that is able to guarantee the desired error masking at the network outputs. In other words, masking will be only partial: the degree of masking inability can be evaluated by measuring the distance between the constraining value and the maximum value of the actual errors.

The most important result of the analysis is in fact that neural networks neither can be considered *completely* intrinsically fault tolerant nor are enhanced learning techniques able to grant *complete* masking ability, with respect to any class of realistic behavioral errors induced by realistic faults.

While all the above has been developed with reference to purely behavioral errors defined on an abstract network model, physical faults affecting a VLSI implementation can be mapped onto such errors so that the expected robustness of a given architecture can be estimated. The critical points that would not allow system survival by neural net-oriented criteria can be detected, so as to apply in such instances architecture-oriented solutions.

As methodological design guidelines, it is possible therefore to summarize the analysis performed and the results achieved in this paper in the following design steps to be performed by the neural architecture designer for the envisioned application:

(a) The actual criticality of the application must be first understood exactly by analyzing the requirements. The bound $\bar{\varepsilon}$ for the output errors (due both to the intrinsic limits of the generalization ability and to the possible faults) must be stated precisely.

(b) The neural paradigm and the learning procedure must be selected, according to the characteristics of the application and by exploiting the knowledge available in the literature.

(c) Training and validation data must be collected to perform learning and to test the achieved generalization ability.

(d) Learning must be applied to the uncommitted neural paradigm to tailor it to the application. The residue error is measured by applying the validation data.

(e) The perspective architectural implementation must be chosen according to possible constraints on the realization (e.g., circuit complexity, throughput, and latency).

(f) The fault model (including both the classes of possible faults and their probability distribution in the neural circuit) must be determined from the considered implementation technology.

(g) The behavioral error model must be derived by mapping faults onto neural errors. For each fault type, the corresponding neural error must be identified, as done in Section 5. The probability distributions of the error classes in the neural paradigm must be computed by adding the occurrence probability of the individual faults leading to the same error.

(h) The adopted and configured neural paradigm must be verified for its masking ability with respect to the application's needs, by using either the theoretical framework discussed in Section 2 or the simulation approach of Section 4 applied to the error model obtained above. If the conditions guaranteeing the masking ability are satisfied, the selected neural paradigm is completely intrinsically fault tolerant for the envisioned application and no further strategy needs to be taken into account to satisfy the fault-tolerance application requirements. If the above conditions are not satisfied but the actual residual error appearing at the neuron outputs in the presence of faults is smaller than $\bar{\varepsilon}$ anyway, the error can be still considered masked by the generalization ability. If the actual error exceeds $\bar{\varepsilon}$—at least for some faults—by a quantity that is tolerable by the application, the intrinsic masking is not *complete* but is still suitable. In any of these cases, the design procedure ends here.

(i) If the masking conditions are not satisfied, the designer should check if there is any enhanced learning technique which is able to achieve the desired level of fault tolerance. This can be done by checking if there is a suitable weight adjustment, either by means of the theoretical analysis shown in Section 3 or by the simulation of Section 4. If a solution exists leading to an acceptable neural error, the theoretical weight adjustment or a most suitable learning technique [19] incorporating fault tolerance into the neural paradigm by enhancing the masking ability through generalization must be adopted. Similarly, if the actual residual error exceeds $\bar{\varepsilon}$ by a quantity that is tolerable, the intrinsic masking is acceptable even if not *complete*. Again, in any of these cases, the design procedure ends here since no further hardware support is required to protect the system.

(j) When intrinsic masking is not enough for the criticality of the application, suitable structural supports must be introduced to achieve fault tolerance at the architectural level. To such a purpose, the analysis of the error distribution provides information about the relevance of each error both (in terms of both frequency and magnitude). The architectural-level strategy must be selected in order to deal with these errors, according to their relevance on the neural computation and to the residual error that is acceptable in the neural outputs. One or more techniques could be required to reduce the output error below the maximum acceptable value, encompassing the solutions proposed, e.g., in [7–12]. The adopted architectural approach must also comply with the application constraints on the implementation (e.g., circuit complexity, throughput, latency). Selection can thus be performed by analyzing the costs and the fault-tolerance benefits of each technique for the envisioned neural paradigm and implementing architecture.

By following these phases, the designer can obtain the best solution for his or her application by taking into account contemporaneously and balancing both behavioral-level constraints on the acceptable neural error and architectural-level limits due to physical realizability and performance required by the application.

REFERENCES

1. D. P. Sieviorek and R. S. Swarz, "Reliable Computer Systems: Design and Evaluation," Digital Press, Burlington, MA, 1992.
2. D. B. I. Feltham and W. Maly, Behavioral modeling of physical defects in VLSI neural networks, *in* "Proc. 1990 International Workshop on Defect and Fault Tolerance in VLSI Systems, Grenoble, France, Nov. 1990."
3. D. B. I. Feltham and W. Maly, Limitation to the size of single-chip electronic neural networks, *in* "Proc. IEEE International Conference on WSI, San Francisco, CA, Jan. 1991," pp. 61–67.
4. F. Distante, M. Sami, R. Stefanelli, and G. Storti-Gajani, Fault tolerant characteristics of the linear array architecture for WSI implementation of neural nets, *in* "Proc. IEEE International Conference on WSI, San Francisco, CA, Jan. 1991," pp. 113–119.
5. V. Piuri, M. Sami, and D. Sciuto, Testability of artificial neural networks: a behavioral approach, *J. Electron. Testing Theory Appl.* **6** (1995), 179–190.
6. C. Alippi, F. Fummi, V. Piuri, M. Sami, and D. Sciuto, Testability analysis and behavioral testing of the Hopfeld neural paradigm, *IEEE Trans. VLSI Systems* in press.
7. V. Piuri, M. Sami, and R. Stefanelli, Arithmetic codes for concurrent error detection in artificial neural networks: the case of AN + B codes, *in* "IEEE Proc. International Workshop on Defect and Fault Tolerance in VLSI Systems, Dallas, TX, Nov. 1992," pp. 277–286.
8. V. Piuri and M. Villa, Residue codes for concurrent error detection in artificial neural networks, *in* "Proc. IEEE-INNS International Joint Conference on Neural Networks, Portland, OR, July 1993," pp. IV.825–IV.830.
9. S. Bettola and V. Piuri, High performance digital neural networks: the use of redundant binary representation for concurrent error detection, *IEEE Trans. Comput.* **47** (March 1998), 357–363.
10. L. Breveglieri and V. Piuri, Error detection in digital neural networks: an algorithm-based approach for inner product protection, *in* "Proc. 1994 SPIE Conference–Advance Signal Processing, San Diego, CA, July 1994," pp. 809–820.
11. Y.-M. Hsu, V. Piuri, and E. E. Swartzlander, Jr., Time-redundant multiple computation for fault-tolerant digital neural networks, *in* "IEEE Proc. International Symposium on Circuits and Systems, Seattle, WA, April 1995," pp. II.977–II.980.
12. Y.-M. Hsu, V. Piuri, and E. E. Swartzlander, Jr., Recomputing by operand exchanging: a time redundancy approach for fault-tolerant neural networks, *in* "IEEE Proc. International Conference on Application-Specific Array Processors, Strasbourg, France, July, 1995," pp. 54–65.
13. M. Stevenson, R. Winter, and B. Widrow, Sensitivity of feedforward neural networks to weight errors, *IEEE Trans. Neural Networks* **1** (March 1990), 71–80.

14. C. Alippi, V. Piuri, and M. Sami, Sensitivity to errors in artificial neural networks: a behavioral approach, *IEEE Trans. Circuits Systems I Fund. Theory Appl.* **42** (1995), 358–361.
15. C. Sequin and R. Clay, Fault tolerance in artificial neural networks, *in* "Proc. IEEE-INNS International Joint Conference on Neural Networks, San Diego, CA, June 1990," pp. 703–708.
16. C.-T. Chiu, K. Mehrotra, K. Chilukuri, and S. Ranka, Modifying training algorithms for improved fault tolerance, *in* "Proc. IEEE International Conference on Neural Networks, Orlando, FL, June 1994," Vol. 1, pp. 333–338.
17. C. Alippi, "Fault Tolerance Analysis in Back-Propagation Neural Networks," Research Note, RM/91/50, Dept. of Computer Science, University College London, 1991.
18. C. Neti, M. H. Schneider, and E. D. Young, Maximally fault tolerant neural networks, *IEEE Trans. Neural Networks* **3** (1992), 14–23.
19. Y. Tan and T. Nanya, Fault-tolerant back-propagation model and its generalization ability, *in* "Proc. IEEE-INNS International Joint Conference on Neural Networks, Nagoya, Japan, Oct. 1993," pp. 2516–2519.
20. D. S. Phatak and I. Koren, Fault tolerance of feedforward neural nets for classification tasks, *in* "Proc. IEEE International Joint Conference on Neural Networks, Baltimore, MD, June 1992," pp. II.386–II.391.
21. V. Piuri, M. Sami, and R. Stefanelli, Neural networks on silicon: the mapping of hardware faults onto behavioral errors, *in* "Proc. International Workshop on Microelectronics for Neural Networks 1991, Munchen, Germany, Oct. 1991," pp. 17–25.
22. V. Piuri, M. Sami, and R. Stefanelli, Fault tolerance in neural networks: theoretical analysis and simulation results, *in* "Proc. Compeuro 1991, Bologna, italy, May 1991," pp. 429–436.
23. D. E. Rumelhart and J. L. McClelland, Eds., "Parallel Distributed Processing: Explorations in the Microstructure of Cognition," Vol. 1, MIT Press, Cambridge, MA, 1986.
24. F. S. Hillier and G. J. Lieberman, "Introduction to Operations Research," Holden-Day, Oakland, 1986.
25. J. Hertz and A. Krogh, "Introduction to the Theory of Neural Computation," Wesley-Addison, Reading, MA, 1991.
26. D. Zhang, G. A. Jullien, W. C. Miller, and E. E. Swartzlander, Jr., Arithmetic for digital neural networks, *in* "Proc. 10th IEEE Symposium Computer Arithmetics, Grenoble, 1991."
27. C. Mead, "Analog VLSI and Neural Systems," Addison-Wesley, Reading, MA, 1989.
28. S. Y. Kung, Parallel architectures for artificial neural nets, *in* "Proc. Systolic Arrays 1988, San Diego, CA, 1988."
29. C. Lehmann and F. Blayo, A VLSI implementation of a generic systolic synaptic building block for neural networks, *in* "Proc. International Workshop on VLSI for Artificial Intelligence and Neural Networks, Oxford, UK, 1990."
30. F. Distante, M. Sami, R. Stefanelli, and G. Storti-Gajani, A compact and fast silicon implementation for layered neural nets, *in* "Proc. IFIP Workshop on Silicon Architectures for Neural Networks, St. Paul de Vence, France, 1990."
31. J. Beichter and U. Ramacher, VLSI design of a neural signal processor, *in* "Proc. IFIP Workshop on Silicon Architectures for Neural Networks, St. Paul de Vence, France, 1990."
32. P. Y. Alla and G. Saucier, Silicon integration of learning algorithms and other autoadaptive properties in a digital feedback neural network, *in* "Proc. IFIP Workshop on Silicon Architectures for Neural Nets, 1990."
33. M. Yasunaga *et al*., Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed of 576 digital neurons, *in* "Proc. IEEE-INNS International Joint Conference on Neural Networks, San Diego, CA, 1990," pp. II.527–II.535.
34. P. Y. Alla, G. Dreyfus, J. D. Gascuel, A. Johannet, L. Personnaz, J. Roman, and M. Weinfield, Silicon integration of learning algorithm and other auto-adaptive properties in a digital feedback neural network, *in* "Proc. IEEE/ITG Workshop on Microelectronics for Neural Networks, Dortmund, Germany, 1990."

35. R. Negrini, M. Sami, and R. Stefanelli, “Fault-Tolerance through Reconfiguration in VLSI Processing Arrays,” MIT Press, Cambridge, MA, 1989.

36. F. Distante, M. Sami, R. Stefanelli, and G. Storti-Gajani, Mapping neural nets onto a massively parallel architecture: a defect tolerance solution, *in* “Proceedings of the IEEE,” **79** (1991), 444–460.