Transactions Briefs

Sensitivity to Errors in Artificial Neural Networks: A Behavioral Approach

Cesare Alippi, Vincenzo Piuri, and Mariagiovanna Sami

Abstract—**The problem of sensitivity to errors in artificial neural networks is discussed here considering an abstract model of the network and the errors that can affect a neuron's computation. Feed-forward multilayered networks are considered; the performance taken into account with respect to error sensitivity is their classification capacity. The final aim is evaluation of the probability that a single neuron's error will affect both its own classification capacity and that of the whole network. A geometrical representation of the neural computation is adopted as the basis for such evaluation. Probability of error propagation is evaluated with respect to the single neuron's output as well as to the complete network's output. The information derived is used to evaluate, for a specific digital network architecture, the most critical sections of the implementation as far as reliability is concerned and thus to point out candidates for ad-hoc fault-tolerance policies.**

I. Introduction

The increasing interest in neural networks has been accompanied by a number of studies concerning fault-tolerance aspects both with respect to the abstract neural paradigms and in relation with specific implementations [1]–[3]. The first of these two approaches is particularly important in the initial phases of design in order to evaluate the intrinsic sensitivity to errors of a chosen neural paradigm and eventually to guide in the development of the neural architecture. Such analysis, being performed at a high abstraction level, implies definition of purely behavioral errors and avoidance of any implementation or technological issues.

Sensitivity to particular classes of errors for specific neural paradigms has been studied, e.g., in [4]–[7]. The authors of [5] and [6] restrict their analysis to errors in synaptic weights or in input signals, considering such errors equivalent as far as their effect on the receiving neuron is concerned, and evaluating the consequences on the behavior of both a single neuron and a complete feed-forward multi-layered network subject to some initial constraints. In particular, limitations on error magnitude are adopted, allowing linearization of the functions involved in evaluation of error propagation probability, and thus, making the underlying mathematics more amenable. All such papers consider only networks upon which learning has been perfected; the same assumption is made in the present paper.

The approach presented in [6] (related to Many-Adalines or Madalines) is the starting point for the more general analysis discussed in this paper. While still considering only multi-layered feed-forward networks, we take into account multiple-step activation functions allowing arbitrary approximations of continuous non-linear functions. We envision possibility of errors in all the abstract operators that constitute the neural computation, including the activation function. (The model is detailed in Section II.) The mathematical frame allowing to evaluate the error probability both at the single neuron's output and at the whole network's outputs is developed in Section III. The complete analysis is derived here for single-step functions, and performances are compared with the ones obtained for a particular instance in [6].

Manuscript received December 17, 1993; revised July 22, 1994. This paper was recommended by Associate Editor C. Jutten.

The authors are with the Department of Electronics and Information, Politecnico di Milano, Milan, Italy.

IEEE Log Number 9411347.

II. Geometrical Models for Computation and Errors

In [6], a geometrical interpretation of operations of an Adaline having input vector $\boldsymbol{X} = \{x_0, \cdots, x_{n-1}\}$ and weight matrix $\boldsymbol{W}$ is given. The set of input vectors in the n-dimensional hyperspace creates a hypersphere that is separated into hemi-hyperspheres by the hyperplane $\boldsymbol{W} \cdot \boldsymbol{X} = 0$. (The extension required in the case of bias is trivial.)

We generalize the model by considering multiple-step symmetric activation functions. Each discontinuity point of the function introduces a hyperplane separating the hyperspace in regions each of which is associated with an output value. More specifically, the locus of the input and output vectors lies on concentric hyperspheres whose number depends on the number of steps into which the function is decomposed.[1] Continuous functions are considered as the limit instance for infinitesimal amplitude of the step-wise interpolation, the input locus being modified into a hypercube.

A comprehensive error model related to the neuron's computation and abstracting from any implementation detail includes the following classes:

1) *input errors*;
2) *weight errors* (weights being fixed, these errors correspond to unexpected weight values);
3) *synaptic product errors*;
4) *summation errors*;
5) *evaluation function errors*.

For error classes 1, 3, 4, and 5, a realistic error assumption is that an unexpected result will be produced as a consequence of the error for at least one value of the variables. A *single error* assumption will be adopted, meaning that a single operator is error-stricken.

A geometrical interpretation of the errors' effects on the neuron's output can be obtained. Consider first weight errors; the error-affected weight vector is $\boldsymbol{We} = \boldsymbol{W} + \boldsymbol{\Delta W}$, being $\boldsymbol{\Delta W}$ the absolute error vector. To observe the relative influence of errors on the computation, the *weight error ratio* $\delta W = |\boldsymbol{\Delta W}|/|\boldsymbol{W}|$ is considered; no constraint is introduced on the magnitude of δW, contrary to the assumption of very small relative errors in [6].

Denote by θ the angle between $\boldsymbol{W}$ and $\boldsymbol{We}$. The classification hyperplanes in the presence of error are rotated with respect to the nominal ones by the same angle θ around a hyperline orthogonal in the origin to the hyperplane identified by $\boldsymbol{W}$ and $\boldsymbol{We}$ [11].

If no bias is considered, in the case of single-step activation functions, this rotation points out two lunes centered in the origin and having angular amplitude θ on the hyperspheres. If a bias is adopted, only one lune is identified on the corresponding hemi-hypersphere.

[1] Consider, for instance, a two-step function where inputs assume the -1, 0, 1 values. The locus of the X vector lies on three concentric hyperspheres of radius 0, 1, $n^{1/2}$.

The output's errors are due to misclassification of the input vectors lying in the lunes.

For a k-step activation function, the rotation affects all hyperplanes separating the regions characterized by different output values. Each hyperplane is rotated by the angle θ around the corresponding rotation axis; it is possible to prove [11] that all rotation axes belong to the same rotation hyperplane orthogonal to the hyperplane containing $\boldsymbol{We}$ and $\boldsymbol{W}$ and passing through the origin. Besides creation of lunes, hyperspherical rings may be created whenever the rotation axis does not intersect the hypersphere, but the hyperplanes cut the hypersphere itself.

When k tends to infinite, the number of the lunes (rings) inside which misclassification occurs tends to infinite as well. As a consequence, all input vectors are misclassified with the exception of those lying in the rotation hyperplane ρ.

Consider now errors affecting the input vector; $\boldsymbol{X}$ is replaced by $\boldsymbol{Xe} = \boldsymbol{X} + \boldsymbol{\Delta X}$. The *input error ratio* is $\delta X = |\boldsymbol{\Delta X}|/|\boldsymbol{X}|$. It can be easily proved that such error induces a rotation of the classifying hyperplane by an angle θ equal to the one between $\boldsymbol{X}$ and $\boldsymbol{Xe}$.

Errors affecting a synaptic product p generate a $p_e = p + \Delta p$, the *product error ratio* being $\delta p = |\Delta p|/|p|$. The characteristics of Δp (in particular, its dependency on inputs) are strongly dependent on the technological implementation of the multiplier.[2]

Errors modifying the expected summation value $\sigma = \Sigma_{i=0}^{n-1} w_i x_i$ lead to $\sigma_e = \sigma + \Delta\sigma$, with $\delta\sigma = |\Delta\sigma|/|\sigma|$. Hyperplanes are moved (not rotated) in the hyperspace; collapsing of two or more hyperplanes is a limit instance of such behavior.

When errors in the activation function are considered, the expected function value f becomes the error-affected value $f_e = f + \Delta f$, the *function error ratio* being $\delta f = |\Delta f|/|f|$. As for product errors, the characteristics of activation function errors and the consequent impact on the neural computation strongly depend on the implementation, i.e., on the actual faults causing the error. For digital implementations, a fault may well cause the function to evaluate outside its legal output codes; non-code words propagation through the network can be predicted only based on the specific implementation.

In the general case, misclassification of the input vectors resulting from activation function's errors is not due to rotation of the hyperplanes separating the classification regions, but rather to regions' modifications induced by different mappings of σ onto output values. Instances related to digital implementations can be listed as *1) change of the position of the hyperplanes, when a discontinuity point is moved; 2) elimination (collapsing) of some hyperplanes; 3) creation of new hyperplanes; 4) change of the output value associated with a region.*

III. Probability of Error Propagation

We denote by *neuron's error probability* and *network's error probability* the probabilities that the error in an entity of the neural computation appears at the neuron's output or at the network's outputs, respectively.

A. The Neuron's Error Probability

The probability of a neuron's output error due to a weight error $P_w(\boldsymbol{\Delta W})$ is defined as the ratio between the number of input vectors misclassified by the neuron and the cardinality of the input space. To evaluate this probability, we refer to the geometric interpretation by considering only the zones Z_h of the hyperspheres mapped by the bounded input values. Within each Z_h, the distribution of the input vectors is uniform or can be assumed uniform whenever the input space dimension is large [8], [9].

$P_w(\boldsymbol{\Delta W})$ is given by the ratio between the area of all lunes and rings generated by hyperplanes' rotation and the area of all hyperspherical zones Z_h. If overlapping of lunes and rings occurs, each hyperspherical surface must be accounted only once. It is

$$P_w(\boldsymbol{\Delta W}) = \frac{\sum_h S\left(\bigcup_k L_{h,k} \cap Z_h\right)}{\sum_h S(Z_h)} \tag{1}$$

where $h \in [1, H]$ is the index identifying the hypersphere, $k \in [1, K_h]$ is the index of each lune or ring $L_{h,k}$ on the hth hypersphere, $S(\cdot)$ is the area of the considered hypersurface, and $\cup_k L_{h,k}$ denotes the hypersurface obtained by the union of all lunes and rings on Z_h.

Since θ is a random variable, and we are interested in its average, we replace θ with the expectation $E(\theta)$ (as in [6]). We express θ as a function of δW by applying trigonometric relationships

$$\theta = \operatorname{atg}\left(\frac{|\boldsymbol{\Delta W}| \cdot \sin\phi}{|\boldsymbol{W}| + |\boldsymbol{\Delta W}| \cdot \cos\phi}\right) = \operatorname{atg}\left(\frac{\delta W \cdot \sin\phi}{1 + \delta W \cdot \cos\phi}\right) \tag{2}$$

where ϕ is the angle between $\boldsymbol{\Delta W}$ and $\boldsymbol{W}$. Equation (2) can be used to obtain $P_w(\delta W)$.

To complete the computation for a specific case and validate our results by reference to those presented in [6], consider a single-step activation function. In this case, there are only one hypersphere and two lunes. Since the lunes are centered in the origin, the ratio between the area of the lunes and the area of the zones is equal to the ratio between the rotation angle θ and π. By replacing θ with its expected value, we obtain

$$P_w(\delta W) = \frac{E(\theta)}{\pi} = \frac{1}{\pi}\int_0^{\pi} \operatorname{atg}\left(\frac{\delta W \cdot \sin\phi}{1 + \delta W \cdot \cos\phi}\right) \cdot p(\phi)\, d\phi \tag{3}$$

where $p(\phi)$ is the probability density function of the random variable ϕ, defined as in [10]

$$p(\phi) = \frac{K_n}{K_{n+1}} \cdot \sin^{n-1}\phi \tag{4}$$

being $K_n = 2\pi^{n/2}/\Gamma(n/2)$, and $\Gamma(\cdot)$ the Gamma function. Integration of (3) (mathematical details are given in [11], [12]) leads to

$$P_w(\delta W) = \frac{1}{2\pi}\left[\frac{\pi}{2} + 2 \cdot \operatorname{atg}\left(\frac{\delta W - 1}{\delta W + 1}\right)\right] \tag{5}$$

for any value of δW. $P_w(\delta W)$ is plotted in Fig. 1 compared with the one given in [6]. For very large values of δW, $P_w(\delta W) \to 1/2$ (as could be intuitively expected given a single step function and uniform input distribution). Conversely, for very small values of δW, the results in [6] are obtained. Consider now input errors. $P_i(\boldsymbol{\Delta X})$ can be computed by referring to a generic $\boldsymbol{W}$. Rotation θ induced by the error may be viewed as a rotation of the coordinate reference system around the origin for an angle equal to $-\theta$. This implies that we can interpret the error on $\boldsymbol{X}$ as an error $\boldsymbol{We}$ on $\boldsymbol{W}$ by making the appropriate substitutions.

Equivalence of these views is granted when $\boldsymbol{We} \cdot \boldsymbol{X} = \boldsymbol{W} \cdot \boldsymbol{Xe}$. To identify the equivalent weight error, we look for a solution $\boldsymbol{We}$, where $\boldsymbol{X}, \boldsymbol{W}$, and $\boldsymbol{Xe}$ are known.

By denoting with μ the angle between $\boldsymbol{\Delta W}$ and $\boldsymbol{X}$, and ν the angle between $\boldsymbol{W}$ and $\boldsymbol{\Delta X}$ (for non null values of $|\boldsymbol{W}|$ and $|\boldsymbol{X}|$), we obtain

$$\delta W = \frac{|\boldsymbol{We} - \boldsymbol{W}|}{|\boldsymbol{W}|} = \frac{|\boldsymbol{Xe} - \boldsymbol{X}|}{|\boldsymbol{X}|} \cdot \frac{\cos\nu}{\cos\mu} = \delta X \cdot \frac{\cos\nu}{\cos\mu}. \tag{6}$$

[2] In the case of analog implementation, where weights are represented by resistors and inputs are given by voltage signals, a multiplication error is, for all practical purposes, equivalent to an error of the corresponding weight.

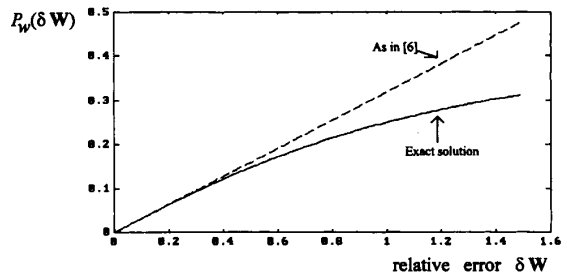

Fig. 1. The probability $P_w(\delta W)$ for the case of single-step nonlinear function.

This equation has an infinite number of solutions. We can safely assume $\cos\nu = \cos\mu$ by which we finally obtain $\delta W = \delta X$. We derive for the case of a single-step function

$$P_x(\delta X) = \frac{1}{2\pi}\left[\frac{\pi}{2} + 2 \cdot \operatorname{atg}\left(\frac{\delta X - 1}{\delta X + 1}\right)\right]. \tag{7}$$

As in the case of weight errors, for small values of δX, this provides the same results given in [6].

Treatment of errors affecting a synaptic product can be reduced to that of weight errors provided a formal correspondence between the product error and an "equivalent" weight error is defined. To this end, let $p_z = W_z X_z$ be the affected product; the product error ratio δp_z is $\delta p_z = |\Delta p_z|/|p_z|$. The equivalent weight error ratio δW is derived as

$$\delta W = \frac{|\Delta p_z|}{|X_z|} \cdot \frac{1}{|\boldsymbol{W}|} = \delta p_z \cdot \frac{|p_z|}{|X_z| \cdot |\boldsymbol{W}|} = \delta p_z \cdot \frac{|W_z|}{|\boldsymbol{W}|}. \tag{8}$$

Consider now a summation error $\Delta\sigma$ shifting the hyperplanes by a segment $\Delta\sigma$. The misclassified input vectors belong to the volume of the hyperspheric segment limited on the hypersphere by the correct and the shifted hyperplanes. $P_\sigma(\Delta\sigma)$ is therefore the ratio between the volume of such a hyperspheric segment and the volume of the whole hypersphere. From $P_\sigma(\Delta\sigma)$, it is possible to derive the expression for probability $P_\sigma(\delta\sigma)$ of observing the relative error $\delta\sigma$ at the neuron's output.

In the case of a single-step evaluation function, for any $\overline{\sigma}$, the probability $P_\sigma(\delta\sigma, \overline{\sigma})$ of observing the relative error $\delta\sigma$ is

$$P_\sigma(\delta\sigma, \overline{\sigma}) = \begin{cases} 0 & \text{if } \delta\sigma \le 1, \text{ i.e., } |\Delta\sigma| \le |\overline{\sigma}|, \\ \frac{1}{2} & \text{if } \delta\sigma > 1, \text{ i.e., } |\Delta\sigma| > |\overline{\sigma}|. \end{cases} \tag{9}$$

The formulation for $P_\sigma(\delta\sigma)$ is derived by integrating the above over all values of $\overline{\sigma}$:

$$\begin{aligned} P_\sigma(\delta\sigma) &= \int_{-\infty}^{+\infty} P_\sigma(\delta\sigma, \zeta) p(\zeta)\, d\zeta \\ &= \begin{cases} 0 & \text{if } \delta\sigma \le 1, \\ \frac{1}{2}\int_{-\infty}^{+\infty} p(\zeta)\, d\zeta = \frac{1}{2} & \text{if } \delta\sigma > 1 \end{cases} \end{aligned} \tag{10}$$

where ζ identifies the generic value $\overline{\sigma}$, and $p(\zeta)$ is the related probability density function for the value ζ.

Finally, probabilistic analysis of activation function's errors is trivial. From definition of neuron's error probability, we are concerned with the evaluation of the probability that an error occurred in the component computing the activation function appears at the neuron's output; we are not interested in the probability of detecting a *fault* by observing its effect onto the computation at the neuron's output.

A fault in the component generating the activation function may be masked in the neuron's output by the implementation of the component itself or by the actual values of inputs and weights. Conversely, an error in the result computed by such a component is always observed at the neuron's output since the component output *is* the neuron's output. Therefore, the neuron's error probability $P_f(\delta f)$ of observing an error δf is always equal to 1.

B. The Network's Error Probability

To observe the error's influence on the computation of the complete multi-layered neural network, it is necessary to propagate the effects of the error from the neuron in which it occurs towards the final outputs. We evaluate then the *network's error probability* $P^*(\delta E)$, i.e., the probability that error $\boldsymbol{E}$ affects the whole computation.

Any single error propagating to the output of a neuron in layer $f (1 \le f < L)$ creates an error on one input of each neuron in layer $f + 1$. We can therefore envision evaluating the probability density function $p(\delta O)$ that a relative error of magnitude δX on the input of any neuron in layer $f + 1$ (characterized by a probability density function $p(\delta X)$) will cause an error δO at the output of the neuron itself. This procedure holds for all neurons in layer $f + 1$; in turn, neurons in layer $f + 2$ will receive inputs with error δX whose probability density function has by now been evaluated, and so on until the network's outputs are reached.

Let $p_l(\delta X)$ be the density function of the probabilistic distribution of the input error ratio δX in each layer l $(f + 1 \le l \le L)$. For each δX, this function gives the probability that δX occurs at the neuron's inputs. The neural computation performed by layer l transforms the layer's input vector $\boldsymbol{X}_l$ into the layer's output vector $\boldsymbol{O}_l$ and the (possible) input error $\boldsymbol{\Delta X}_l$ into the output error $\boldsymbol{\Delta O}_l$. Let δO_l be the output error ratio ($\delta O_l = |\boldsymbol{\Delta O}_l|/|\boldsymbol{O}_l|$). The probability density $p_l(\delta X_l)$ is transformed by the neural computation into the corresponding probability density function $p_l(\delta O_l)$. Such function gives the probability that a non-null δO_l appears at the layer's outputs.

In the present case, we need to evaluate the probability density function $p(\delta O)$, not simply the probability of observing an error. Therefore, given δX, and the input-output mapping implemented by the neuron, we must identify the possible values of δO and the probability density function associated with such values. To this end, we rely on the geometric interpretation adopted in the previous sections.

Let $p_{\delta O_l|\delta X_l}$ be the conditional probability that a relative output error δO_l is generated whenever the input relative error δX_l is present. The probability density function $p_l(\delta O_l)$ is obtained by evaluating the expected value of the conditional probability

$$p_l(\delta O_l) = \sum_{\delta X_l} p_{\delta O_l|\delta X_l} \cdot p(\delta X_l). \tag{11}$$

Consider as a working example neurons characterized by a single-step activation function. For any value δX, the value of δO is either 0 or 2; in fact, the output error $\boldsymbol{\Delta O}$ of a single-step activation function—if any—is either +2 or −2, while $|\boldsymbol{O}| = 1$. The conditional probability of observing the error related to each particular value of δX is $p_{\delta O_l|\delta X_l} = P_x(\delta X_l) \cdot \delta(2)$, where $\delta(2)$ is the Dirac's function centered in δO =2. The total probability density function $p(\delta O_l)$ is determined as $p(\delta O_l) = \delta(2) \cdot \Sigma_{\delta X} P_x(\delta X_l) \cdot p(\delta X_l)$. By iterating the above procedure, we obtain the probability density functions for each

TABLE I
A SAMPLE EVALUATION OF THE ERROR PROBABILITY IN THE PRESENCE OF PRODUCTION-TIME DEFECTS

Defect Type	Defect Number	Defect Probability	Error Propagation Probability	Output Error Probability
hidden neuron's memory	0.6670	0.2081	0.2478	0.0516
output neuron's memory	0.0334	0.0105	0.5033	0.0168
hidden neuron's multiplier	0.4104	0.1280	0.2480	0.0317
output neuron's multiplier	0.0205	0.0064	0.5017	0.0032
hidden neuron's adder	1.9755	0.6163	0.0267	0.0165
output neuron's adder	0.0987	0.0307	0.1171	0.0036

neuron in the output layer. Integration of each such function over the whole range of possible values of the output error ratio provides the probability of observing an error at the output of the corresponding neuron. To have a first approximation of the network's sensitivity, the maximum value of all probabilities can be considered; in turn, it provides a lower bound for the error probability $P^*(\delta E)$.

IV. CONCLUDING REMARKS

Equations derived in this paper can be used as guidelines during design of artificial neural networks. To provide an example of such a procedure, we chose a simple three-layered network composed of 20 inputs, 15 neurons in the hidden layer, and one output neuron, trained to detect the presence of a ship in a radar image. The digital architecture adopted for the neurons is derived from that suggested in [13], where each neural operators is directly mapped onto a corresponding device, and the activation function is the single-step one. A preliminary VLSI design has been carried out, adopting a 0.7 μm CMOS technology; 8 b have been chosen for weight representation and 13 b for the adders (to avoid truncation problems). A defect distribution of 15 defects per square centimeter has been assumed (since we are evaluating the presence of production-time defect) as a reasonable parameter provided by recent literature.

As a consequence, the information summarized in Table I has been derived. For different classes of defects, we evaluated the total number of defects in each class as related to the silicon area occupied by the digital devices, the relative occurrence probability of each defect class, the probability that an error (due to the defects) propagates to the network's output, and finally, the probability of observing an error in the network's output due to the defect distribution. (This probability is the product of the defect occurrence probability by the error propagation probability.)

Simple analysis of this table allows us to identify the weight memory of the hidden layer as the most critical section in the design. It is interesting to note that also the multiplier section of the hidden layer—however simple its design—has a fairly relevant influence on the network's ouput. As a consequence, fault-tolerance policies (or, at least, concurrent fault-detection techniques) should be concentrated in these two subsystems.

REFERENCES

[1] C. Neti, M. H. Schneider, and E. D. Young, "Maximally fault tolerant neural networks," *IEEE Trans. Neural Networks*, vol. 3, no. 1, pp. 14–23, Jan. 1992.
[2] D. B. I. Feltham and W. Maly, "Behavioral modeling of physical defects in VLSI neural networks," in *Proc. Int. Workshop on Defect and Fault Tolerance in VLSI Systems*, Grenoble, Nov. 1990.
[3] F. Distante, M. Sami, R. Stefanelli, and G. Storti-Gajani, "Mapping neural nets onto a massively parallel architecture: A defect-tolerance solution," *Proc. IEEE*, vol. 79, no. 4, pp. 444–460, Apr. 1991.
[4] V. Piuri, M. Sami, and R. Stefanelli, "Fault tolerance in neural networks: Theoretical analysis and simulation results," in *Proc. Compeuro 1991*, Bologna, Italy, 1991.
[5] L. A. Belfore, II and B. W. Johnson, "The analysis of the faulty behavior of synchronous neural networks," *IEEE Trans. Comput.*, vol. 49, no. 12, pp. 1424–1429, Dec. 1991.
[6] M. Stevenson, R. Winter, and B. Widrow, "Sensitivity of feedforward neural networks to weight errors," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 81–92, Mar. 1990.
[7] C. Alippi, "Asymptotic insensitivity to weights perturbations in back-propagation classifiers," in *Proc IJCNN93*, Nagoya, Japan, Oct. 1993.
[8] M. E. Hoff, Jr., "Learning phenomena in networks of adaptive switching circuits," Ph.D. dissertation, Dept. Electrical Eng., Stanford Univ., Stanford, CA, June 1962.
[9] F. H. Glanz, "Statistical extrapolation in certain adaptive pattern recognition systems," Ph.D. dissertation, Dept. Electrical Eng., Stanford Univ., Stanford, CA, May 1965.
[10] R. G. Winter, "Madaline rule: A new method for training networks of Adalines," Ph.D. dissertation, Dept. Electrical Eng., Stanford Univ., Stanford, CA, Jan. 1989.
[11] C. Alippi, V. Piuri, and M.G. Sami, "The issue of error sensitivity in neural networks," in *Proc. Int. Conf. on Massively-Parallel Computing Systems MPCS94*, Ischia, Italy, May 1994.
[12] ——, "Sensitivity to errors in artificial neural networks: A behavioral approach," in *Proc. ISCAS'94*, London, UK, June 1994.
[13] C. Lehmann and F. Blayo, "A VLSI implementation of a generic systolic synaptic building block for neural networks," in *Proc. Int. Workshop on VLSI for Artificial Intell. and Neural Networks*, Oxford, UK, 1990.