

The Issue of Error Sensitivity in Neural Networks

Cesare Alippi, Vincenzo Piuri, Mariagiovanna Sami
Department of Electronics and Information
Politecnico di Milano
Milano, Italy

Abstract

The problem of sensitivity to errors in artificial Neural Networks is discussed here in behavioral terms, i.e. considering an abstract model of the network and the errors that can affect a neuron's computation. Feed-forward multi-layered 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 the whole network's classification capacity. 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.

1 Introduction

Neural networks are increasingly seen as viable solutions in application areas requiring massively parallel processing to meet hard real-time constraints when affording problems that are not solved by algorithmic approaches. In this context, the trend is towards implementation of dedicated neural architectures, for which the learning phase is completed off-line, possibly by simulation on a host machine, so that both network topology and weight distribution are completely defined. Such phase actually leads to fix the computational paradigm, i.e., the *abstract architecture* of the neural network, but it does not a priori constrain the real architecture upon which it will be mapped (even less, the implementing technology). Performance re-

quirements and other figures of merit will have to be taken into account to finalize the system's design.

Adoption of neural networks in possibly critical application areas has led to consider the influence of errors on the neural computation, to provide guidelines on fault-tolerance policies required. Two approaches are discernible in present literature. On the one hand, efficient architectures for given applications are designed first, and fault-tolerance measures are then inserted. Actually, in such instances (see [1-4]), the nature of the neural computation is not taken into account, but rather the architectural characteristics constitute the basis for any fault-tolerance policy chosen. This is true in particular of all implementations based on array structures, such as the one discussed in [4].

An alternative approach, adopted by several authors [5-8], deals with the abstract paradigm of the neural network, examining the effects of purely *behavioral* errors, relating to the operators involved in the neural computation rather than with faults in a digital architecture. As a result, probability of a given error's influence upon the global neural computation can be evaluated; in other words, the *sensitivity to errors* of the neural paradigm is evaluated. Such an approach has easily disproved an initial intuitive belief in the intrinsic robustness of the neural paradigm (a robustness that can be achieved only by a proper learning procedure finally resulting in careful distribution and redundancy of information within the network). Moreover, and even more important, its results can provide useful guidelines both in the choice of one implementing architecture among a set of alternatives and, given one architecture, in localization of "critical" subsystems where suitable self-checking and fault-tolerance policies should be concentrated. In this case, a mapping of architectural faults onto behavioral errors should be performed first.

Sensitivity to specific classes of perturbations for given neural paradigms has been studied at different

abstraction levels to determine the influence of errors in the neural computation at the network outputs [5-7, 9-11] and finally to measure the induced loss in final performance or performance distortion [9, 12, 13]. As sensitivity issues are concerned, the authors of [6] and [7] 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 the evaluation of the error propagation probability and thus making the underlying mathematics more amenable. Results, limited to binary inputs and step activation functions, have been partially extended in [9] by considering continuous functions. Under the large fan-in assumption for each artificial neuron (which allows exploitation of the central limit theorem), a stochastic frame is provided to analyse the sensitivity of a neural network's outputs to behavioral errors affecting weights. This hypothesis has been weakened in [10, 11] to evaluate the probability of observing an error at the output of the neuron affected by the error itself. Performance distortion involves definition of figures of merit to measure the loss in performance due to perturbations affecting the neural parameters. Such figures rely on the analysis of signal and noise: they are generally based either on the ratio of signal-to-noise variances [9] or on the ratio of average powers [12]. In [13] a generalisation of the influence of precision requirements has been performed to deal with all functional units in a dedicated digital neural architecture.

In the present paper, we consider *application-specific* neural systems. By this, we mean that *network topology* (restricted within the multi-layered feed-forward class) as well as *weight distribution* have been completely determined (for example, in a preliminary simulation phase); therefore, the abstract operators taken into account for definition of the error model are all and only the ones involved in the *recall* phase. This is the only restriction introduced as far as implementation is concerned; no details concerning architecture or even technology are required. The approach presented in [7] (related to Many-Adalines or Madalines), and in particular the geometric representation there adopted, are the starting point for the more general analysis discussed in the present paper. Contrary to the limitation in [7], we take into account multiple-step activation functions, accepting as

limit instance (for an infinity of steps) continuous non-linear functions. Extension to multiple-step functions is made necessary by the fact that, whenever three or more layers are present, the single-step function does not grant arbitrarily small errors in the recall phase, unless the initial network topology is modified [14]. We envision possibility of errors in all the abstract operators that constitute the neural computation, including the activation function.

In Section 2, the geometrical description of the neural computation will be introduced, as well as the approach adopted for subsequent error-sensitivity analysis. In Section 3, the error model will be described in detail, starting from behavioral errors associated with the operators appearing in the neural computation. A geometrical representation of errors and of their effects on the neural computation will be discussed; relative errors of arbitrary magnitude are allowed. In Section 4, the probability of propagating the errors to the output of the single neuron will first be evaluated. Detailed evaluation will be carried out for the case of single-step activation functions; the general approach for multi-step and continuous function will be defined. Then, probability of error propagation will be studied with respect to the outputs of a general multi-layered feed-forward network.

2 The Neural Computation and the Geometrical Model

Let us define first the general computational model to which reference will be made in this paper. As already said, *feed-forward multi-layered* networks will be taken into account (a sample scheme is given in fig. 1). The individual neuron is identified by two indices: the layer k and the position i within the layer. Its functionality is defined by [15]:

$$\begin{aligned} x_{k,i} &= f_{k,i}(\sigma_{k,i} - \theta_{k,i}) = \\ &= f_{k,i}\left(\sum_{j=0}^{n_{k-1}-1} w_{k-1,i,j} x_{k-1,j} - \theta_{k,i}\right) \end{aligned} \quad (1)$$

where $x_{k,i}$ is the output of the i -th neuron in layer k (it is therefore the input for neurons in the subsequent layer), $w_{k-1,i,j}$ is the interconnection weight from neuron $(k-1, j)$ to neuron (k, i) , $\sigma_{k,i}$ is the weighted input summation for neuron (k, i) , $\theta_{k,i}$ is the threshold associated to neuron (k, i) , and $f_{k,i}(\cdot)$ is the non-linear activation function. External inputs (typically provided by a set of sensors) are applied to neurons of layer 0

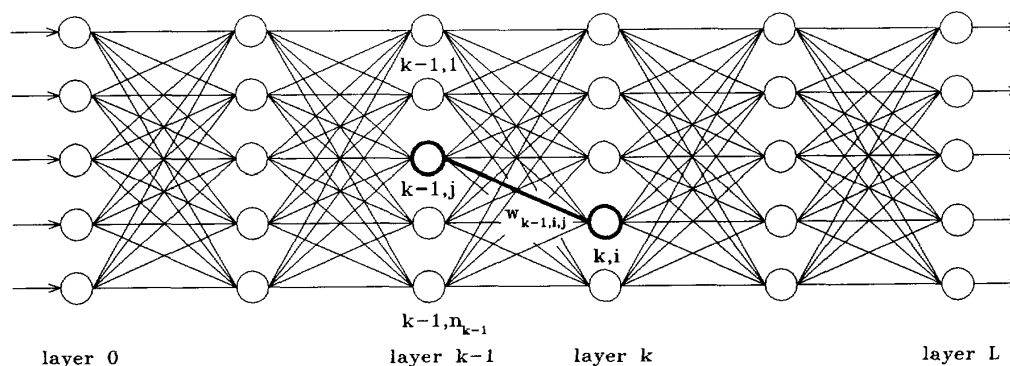


Fig. 1 - Sample feed-forward multi-layered network

(the input layer), each of which receives one input signal. The set of inputs defines the input hyperspace. The learning procedure allows neural networks of the class here considered to partition the input hyperspace into disjoint classes: the neural computation leads to identify the class corresponding to the applied inputs. The final outputs of the network (generated by the output layer) represent the classification of the actual input into the given classes.

In [7] a simplified neural model has been considered, based on the Adaline introduced by Widrow [15], where the activation function is a single-step function. Multi-layered feed-forward structures (called Madalines) are built consisting of Adalines. The output is 1 if $\sigma \geq 0$; otherwise, it is -1 . The presence of a bias ω can be included in the previous σ expression by representing it as the product of a constant input (equal to 1) by the symbolic weight ω ; from now on, therefore, the bias input will be considered as any nominal input.

We can represent the Adaline's operation in the n -dimensional input hyperspace if no bias is applied (otherwise, an additional dimension should be added). In such hyperspace, the set of inputs is represented by a vector $\mathbf{X}$ through the origin, and the set of synaptic weights is represented by a vector $\mathbf{W}$; σ is the scalar product $\mathbf{W} \cdot \mathbf{X}$. Since each individual input has only two possible values, a set of 2^n input vectors can be defined; if no bias is present, the corresponding points in the hyperspace lie on a hypersphere with radius $\sqrt{n}$ centered in the origin. If a bias is present, the points lie on a hemi-hypersphere with radius $\sqrt{n+1}$ centered in the origin and oriented towards positive bias values. The activation function partitions the set of input vectors into disjoint subsets associated each with one

value of the function itself; in the particular instance of an Adaline, the function creates the hyperplane $\mathbf{W} \cdot \mathbf{X} = 0$ as separator between the two subsets of input vectors. This hyperplane is orthogonal to $\mathbf{W}$ in the origin. The hemi-hypersphere lying in the direction of $\mathbf{W}$ corresponds to positive values of σ , thus inducing the value $+1$ of the output.

We introduce now our generalization of this model. For simplicity we consider only symmetric functions (this does not involve any loss of generality in the treatment). We assume, again without any loss of generality, that all inputs range in the same interval of values $[-1, 1]$; this guarantees that, when no bias is considered, the locus of the vectors $\mathbf{X}$ lies on a hypersphere (different ranges lead to a hyperelliptic distribution of points), so that the evaluation of all geometrical characteristics is simplified. If a bias is introduced, the locus lies on hemi-hyperspheres.

Consider first a two-step function whose output may assume three different symmetric values (namely $-1, 0, +1$). The locus of the $\mathbf{X}$ vectors lies on three concentric hyperspheres of radius, respectively, 0, 1 and $\sqrt{n}$, and is partitioned into three regions by the different values of the weighted input summation σ . These regions are separated by two hyperplanes defined respectively by $\mathbf{W} \cdot \mathbf{X} = +a$ and $\mathbf{W} \cdot \mathbf{X} = -a$, where $+a$ and $-a$ are the discontinuity points of the activation function; the hyperplanes are orthogonal to $\mathbf{W}$ and their distance from the origin is $a \cdot |\mathbf{W}|^{-1}$. The region containing the origin corresponds to values between $-a$ and $+a$ and, thus, generates a zero output. The region in the direction of $\mathbf{W}$ contains the $\mathbf{X}$ vectors generating values greater than $+a$ and output equal to 1, while the opposite region corresponds to values smaller than $-a$ and to outputs equal to -1 .

Generalization to the composition of more step functions is now straightforward. A multiple-step function can be seen as the stepwise interpolation of a continuous limited function; by choosing an infinitesimal interpolation step $\delta\sigma$, the continuous function is obtained as an infinite-step interpolation. If no bias is considered, we introduce an infinite number of hyperspheres onto which the $\mathbf{X}$ vectors' locus lies; such hyperspheres are centered in the origin and their radius ranges densely between 0 and $\sqrt{n}$. If a bias is adopted, the locus lies on hemi-hyperspheres in the direction of positive bias. Since there is an infinite number of discontinuity points, there is an infinite number of hyperplanes separating the locus into (infinite) regions. All these hyperplanes are orthogonal to $\mathbf{W}$ and the distance between each adjacent pair becomes infinitesimal. Note that the decomposition of the nonlinear activation function as the sum of an infinite number of infinitesimal step functions leads to consider the inputs x_i and the neuron's output as a dense set. This implies that the $\mathbf{X}$ vectors' locus has a dense structure.

Finally, when the infinitesimal amplitude of the stepwise interpolation arbitrarily tends to zero, then the $\mathbf{X}$ vectors' locus becomes continuous. In the discrete case, the locus does not coincide with the complete set of hyperspheres, but actually is contained within the hypercube having edge equal to 2 and centered in the origin. In the continuous case, the locus coincides with such hypercube.

3 The Error Model and its Geometrical Interpretation

We introduce a behavioral error model to describe the erroneous computation due to the occurrence of faults in the ANN. Errors are related to the abstract variables and operators appearing in the definition of a neuron's computation [6]; an error is simply a non-null difference between the expected value and the actual value. Such a definition does not account for any particular implementation technology or circuit solution; bounds to error magnitude or characteristics may be derived from physical reality.

The following classes of errors may be identified:

1. *input errors*;
2. *weight errors*;
3. *synaptic product errors*;
4. *summation errors*;

5. *evaluation function errors.*

Since weights are fixed, any fault which modifies the weight's value appears as an error. 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. This assumption corresponds to the reality of physical implementations; e.g., a stuck-at fault in a digital multiplier will not affect all the device's results nor affect all of them in the same way.

A geometrical interpretation of the effects caused by the various errors will be presented in the following subsections; errors will be seen as modifications of the computational parameters. A *single error* assumption will be adopted, meaning that a single operator is error-stricken.

We take into account general non-linear limited activation functions, rather than simple step functions; nevertheless, for brevity, we refer to the network's output functionality as to its "classification ability", adopting the same vocabulary used in [7]. Actually, we can refer to a "classification ability" even in the case of multi-step activation functions, classification being now a multi-valued function.

3.1 Geometrical Interpretation of Weight Errors

The weight vector $\mathbf{W}$ and the non-linear activation function determine the input-output mapping of the neuron. A change in $\mathbf{W}$, or in the input vector $\mathbf{X}$, or in the non-linear activation function can alter this mapping and, thus, the network's classification ability.

Consider first weight errors: they can be described by adopting an additive error model (as done also in [7]) so that the error-affected weight vector $\mathbf{W}_e$ is:

$$\mathbf{W}_e = \mathbf{W} + \Delta\mathbf{W} \quad (2)$$

where $\Delta\mathbf{W}$ is the weight error induced in $\mathbf{W}$. No constraints are introduced here on the magnitude of $\Delta\mathbf{W}$. Adoption of an error vector actually allows a multiplicity of errors in the various weight components; this accounts well for physical faults in some digital implementations, where all weights associated with inputs of a given neuron can be corrupted by the presence of even a single physical fault. To abstract from the actual magnitude of these vectors and to observe the relative influence of errors on the computation, the *weight error ratio* $\delta W = |\Delta\mathbf{W}| / |\mathbf{W}|$ is considered.

Refer to the geometrical representation in fig. 2. All hyperplanes describing the classification ability

(i.e., the regions with the same output value) are now orthogonal to the actual weight vector $\mathbf{W}_e$. Denote by θ the angle between $\mathbf{W}$ and $\mathbf{W}_e$. The classification hyperplanes in the presence of error are rotated with respect to the nominal ones by the same angle θ around the hyperline r , which is orthogonal to the hyperplane identified by $\mathbf{W}$ and $\mathbf{W}_e$, in the origin [14]. 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 (see fig. 2). If a bias is adopted, only one lune is identified on the corresponding hemi-hypersphere. The output's errors are due to misclassification of the input vectors lying in the lunes. The analysis performed in [7] involves the restrictive assumption of very small relative weight errors.

If a two-step activation function is considered, the error induces a rotation of the two hyperplanes α and β that separate the regions associated with the different output values (see fig. 3). Also in this case, the hyperplanes' rotation angle is equal to the angle θ between $\mathbf{W}$ and $\mathbf{W}_e$ [14].

Consider pairs of hyperplanes consisting each of a nominal hyperplane (the one in the absence of errors) and of the corresponding hyperplane in the presence of the error. For each hypersphere on which the $\mathbf{X}$ vectors' locus lies, if there is no bias and the rotation axis of a hyperplanes' pair intersects such hypersphere, two lunes are cut on the hypersphere by the hyperplanes. No lunes are generated if the hyperplanes never intersect the hypersphere; one hyperspherical ring is created if the rotation axis does not intersect the hypersphere, but the two hemi-hyperplanes cut the hypersphere itself. If the bias is taken into account and the rotation axis does not intersect the hypersphere, only one lune is drawn onto the hemi-hypersphere containing the $\mathbf{X}$ vectors' locus. No lune is generated if the hyperplanes' pair never intersects the hypersphere; an hyperspheric ring (or a part of a hyperspherical ring) is identified onto the hemi-hypersphere if the rotation axis does not intersect the hypersphere, but the two hemi-hyperplanes cut the hemi-hypersphere. Note that lunes and rings may overlap if the weight error ratio is large enough. The error appears at the neuron's output for $\mathbf{X}$ vectors lying in such lunes and rings.

If a k -step activation function is considered, all hyperplanes separating the regions characterized by different output values are rotated. When k tends to infinite, the number of 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 .

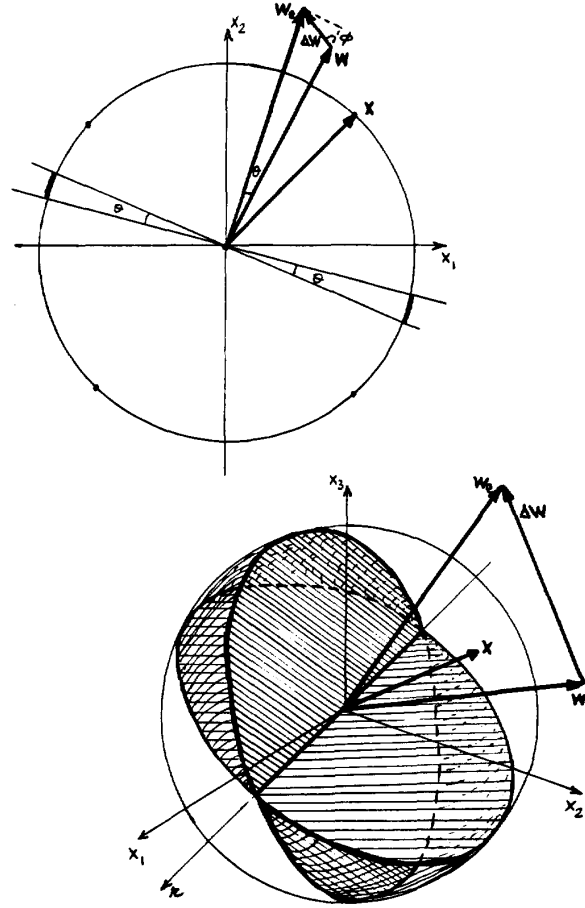


Fig. 2 - Geometric interpretation of a weight error in a single-step function

3.2 Geometrical Interpretation of Input Errors

In this case, the input vector $\mathbf{X}$ is replaced by the error-affected input vector $\mathbf{X}_e$. By adopting the same additive error description, it is:

$$\mathbf{X}_e = \mathbf{X} + \Delta\mathbf{X} \quad (3)$$

where $\Delta\mathbf{X}$ is the input error induced on $\mathbf{X}$. The *input error ratio* is $\delta X = |\Delta\mathbf{X}| / |\mathbf{X}|$.

In fig. 4, an example for a single-step activation function is shown. The hyperplane $\mathbf{W} \cdot \mathbf{X}_e = 0$ describes the classification ability of the neuron in the presence of an input error; it separates the two regions associated with the two output values, and it is

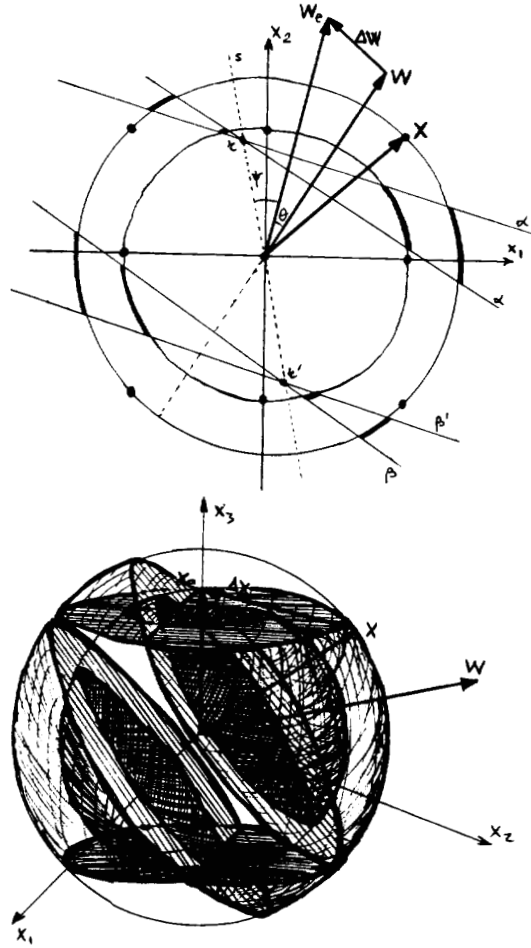


Fig. 3 - Geometric interpretation of weight error in a two-step function

orthogonal to $\mathbf{X}_e$ in the origin. Let δ be the hyperplane identified by $\mathbf{X}$ and $\mathbf{X}_e$, and θ the angle between $\mathbf{X}$ and $\mathbf{X}_e$. The hyperplane $\mathbf{W} \cdot \mathbf{X}_e = 0$ can be obtained by rotating the nominal one ($\mathbf{W} \cdot \mathbf{X} = 0$) of θ around the r hyperline (defined as in Section 3.1). The two hyperplanes determine two lunes on the hypersphere containing the $\mathbf{X}$ vectors' locus, centered in the origin and with angular amplitude equal to θ . Misclassification occurs for all input vectors belonging to these lunes. As for weight errors, the lunes (or rings) may overlap.

These results can be extended to deal with a k -step activation function. All hyperplanes separating the regions with different output values are rotated of θ around the corresponding axis. The pairs of hyper-

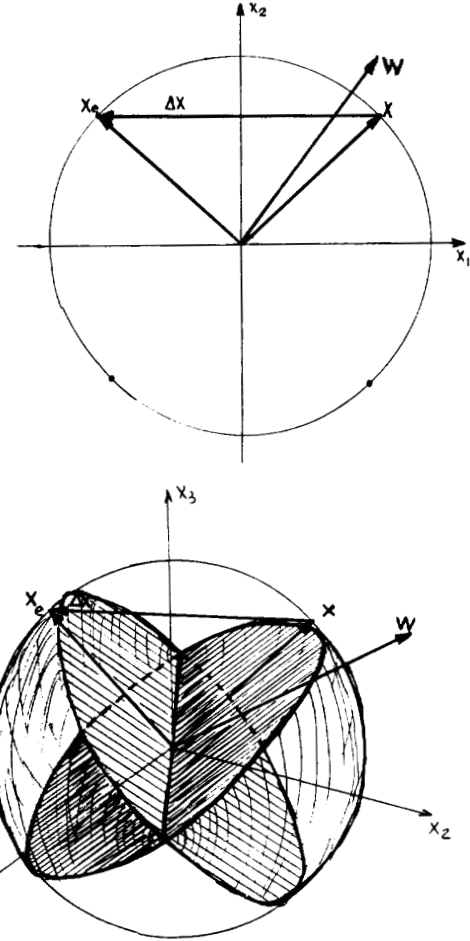


Fig. 4 - Geometric interpretation of input errors in a single-step function

planes associated with the rotation generate at most $2(k-1)$ lunes or hyperspherical rings on each hypersphere on which the $\mathbf{X}$ vectors' locus lies. For increasing values of k , the probability of lunes/rings overlapping is higher. Erroneous classification occurs for all input vectors belonging to these lunes and rings. For k going to infinity and for continuous activation function, all $\mathbf{X}$ vectors are misclassified except those lying onto the rotation hyperplane ρ .

3.3 Geometrical Interpretation of Synaptic Product Errors

The expected synaptic product p is changed into the error-affected product $p_e = p + \Delta p$, where Δp

is the product error. The effective influence on the computation can be characterized by the *product error ratio* $\delta p = |\Delta p| / |p|$.

The outcome of a multiplication error is strongly dependent on the technological implementation of the multiplication itself. 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, so that the analysis performed in Section 3.1 is immediately adopted.

In the case of digital implementation, a detailed analysis of the circuit structure should be performed. Solutions involving time-multiplexing of the multiplier (which would lead to possible multiple product errors induced by a single physical fault) are not examined here, since we restrict ourselves to single neural errors. Otherwise, as a consequence of a product error, the hyperplane $\mathbf{W} \cdot \mathbf{X} = 0$ is rotated by an angle θ .

3.4 Geometrical Interpretation of Summation Errors

These errors modify the expected summation value σ into the error-affected summation $\sigma_e = \sigma + \Delta\sigma$, where $\Delta\sigma$ is the summation error. The effective influence on the neural computation can be characterized by the *summation error ratio* $\delta\sigma = |\Delta\sigma| / |\sigma|$.

Also in this case, the error characteristics depend on the implementation of the operator. In simple analog realizations operands are often current signals input to a grounded node: σ is the overall current flowing to ground. An open circuit on the ground path induces a fixed value (zero) of the summation; this is equivalent to remove all hyperplanes from the geometrical model and to generate always the same result at the neuron output. The same behavior occurs in the case of a serial digital adder when a stuck-at fault affects the output adder. In general, hyperplanes are moved (not rotated) in the hyperspace; collapsing of two or more hyperplanes is a limit instance of such behavior.

3.5 Geometrical Interpretation of Errors in the Activation Function

When errors in the activation function are considered, the expected function value f becomes the error-affected value $f_e = f + \Delta f$, where Δf is the function error. The *function error ratio* can be defined as $\delta f = |\Delta f| / |f|$.

Also in the case of activation function's errors, we cannot avoid reference to the actual implementation to

describe the influence of faults onto the neural computation. A fault, in particular for digital implementations, may well cause the function to evaluate outside its legal output codes; non-codewords propagation through the network can be predicted only based on the specific implementation.

Misclassification of the input vectors 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 due to distortion of the activation function. The shape of such distortions depends on the specific implementation adopted; even though we consider "behavioral" error definitions, instances to which we refer are related to digital implementations and can be listed as follows:

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.

In the first case, the boundary between two adjacent regions is still a hyperplane, but its equation is different from the nominal one. The hyperplanes corresponding to nominal and erroneous discontinuity points select a hyperspherical ring on each hypersphere on which the $\mathbf{X}$ vectors locus lies; misclassification occurs for all $\mathbf{X}$ vectors belonging to such rings. Possible collapsing of two hyperplanes may occur if the error moves them to the same final position, i.e., if it moves the values of σ corresponding to discontinuity points into a unique point. Again, misclassification occurs for $\mathbf{X}$ vectors belonging to the selected rings.

Creation of new discontinuity points in the activation function (case 3) may lead to introduce new hyperplanes and new regions. These hyperplanes are orthogonal to the weight vector, as are the nominal ones. Misclassification occurs for all $\mathbf{X}$ vectors in the rings (or part of rings) identified by the new regions on the hyperspheres on which the $\mathbf{X}$ vectors' locus lies.

Modification of the output value in a region (case 4), without moving the discontinuity points, does not induce any change in the position of the hyperplanes. Since the actual output for all $\mathbf{X}$ vectors of such region is different from the expected one, misclassification occurs for all $\mathbf{X}$ vectors in the rings (or part of them) identified by that region on the hyperspheres on which the $\mathbf{X}$ vectors' locus lies.

4 Probability of Error Propagation

The geometrical interpretation introduced in Section 3 will be used in the present section to compute the probability of error propagation to the outputs of the error-stricken neuron and, afterwards, to the output of the complete neural network. 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. Such information, together with data on the error rate of devices or functional blocks envisioned in a given architecture, will allow to evaluate the robustness of the implementation and, if so required, to introduce appropriate fault tolerance measures.

4.1 The Neuron's Error Probability for Weight Errors

We refer here directly to multi-step activation functions, with an arbitrary number of steps. The probability of a neuron's output error due to a weight error $P_w(\Delta \mathbf{W})$ 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.

Since each input ranges in a bounded interval, we consider only the zones Z_h of these hyperspheres that satisfy this constraints on the input values. Within each such hyperspherical zone, the distribution of the input vectors is uniform or can be assumed uniform when the number of input space dimensions is large [16, 17].

As already said in Section 3.1, misclassification occurs for input vectors belonging to the lunes or to the rings generated by hyperplanes' rotation. Under the uniform distribution assumption for input vectors, $P_w(\Delta \mathbf{W})$ is given by the ratio between the area of all lunes and rings 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:

$$P_w(\Delta \mathbf{W}) = \frac{\sum_h S(\cup_k L_{h,k} \cap Z_h)}{\sum_h S(Z_h)} \quad (4)$$

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 h -th 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 the hyperspherical zone Z_h .

No constraint is imposed on the amplitude of the rotation angle θ , i.e., on the magnitude of the error $\Delta \mathbf{W}$ and on the weight error ratio δW . Since θ is a random variable and we are interested in its average value, we replace the angle θ in the evaluation of δW with its expected value $E(\theta)$ (as in [7]).

Actually, rather than considering $P_w(\Delta \mathbf{W})$, we can refer to probability $P_w(\delta W)$ depending only on the relative magnitude δW of the weight error. By using the trigonometric relationships, we obtain:

$$\begin{aligned} \theta &= \text{atg} \left(\frac{|\Delta \mathbf{W}| \cdot \sin \phi}{|\mathbf{W}| + |\Delta \mathbf{W}| \cdot \cos \phi} \right) = \\ &= \text{atg} \left(\frac{\delta W \cdot \sin \phi}{1 + \delta W \cdot \cos \phi} \right) \end{aligned} \quad (5)$$

where ϕ is the angle between $\Delta \mathbf{W}$ and $\mathbf{W}$.

As an example, 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

$$\begin{aligned} P_w(\delta W) &= \frac{E(\theta)}{\pi} = \\ &= \frac{1}{\pi} \int_0^\pi \text{atg} \left(\frac{\delta W \cdot \sin \phi}{1 + \delta W \cdot \cos \phi} \right) \cdot p(\phi) d\phi \end{aligned} \quad (6)$$

where $p(\phi)$ is the probability density function of the random variable ϕ , defined as in [18].

Integrating equation (6) for the possible different values of δW allows to obtain:

$$P_w(\delta W) = \frac{1}{2\pi} \left[\frac{\pi}{2} + 2 \cdot \text{atg} \left(\frac{\delta W - 1}{\delta W + 1} \right) \right] \quad (7)$$

for any value of δW (the mathematical details are given in [14]). $P_w(\delta W)$ is plotted in fig. 5. It can be noticed that, for very large values of δW , $P_w(\delta W) \rightarrow 1/2$ (as could be intuitively expected given a single step function and uniform input distribution). Conversely, for very small values of δW , Widrow's results [7] are obtained (fig. 6 shows the approximation error of Widrow's approach with respect to our approach).

4.2 The Neuron's Error Probability for Input Errors

Probability $P_i(\mathbf{X}_e)$ for a given $\mathbf{X}_e$ is the ratio between the number of weight vectors that induce misclassification and the total number of weight vectors.

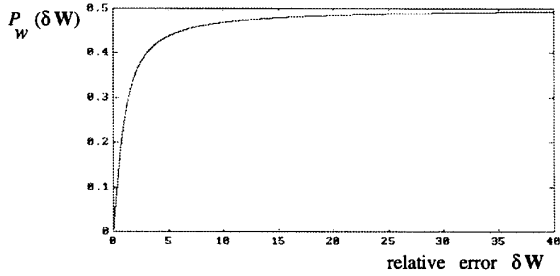
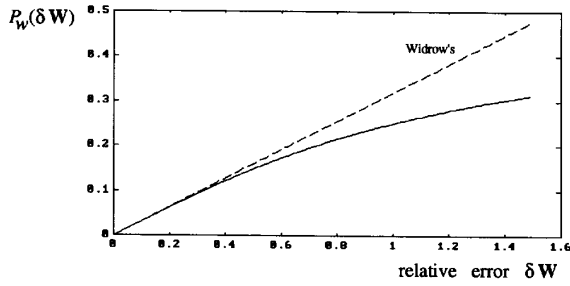


Fig. 5 - The probability $P_w(\delta W)$ for the case of single-step non-linear function

Evaluation of such probability is based on the geometrical interpretation and follows the main guidelines suggested in Section 4.1. However, we will show in this section how such an evaluation is redundant since analysis of input errors may be reduced to analysis of weight errors and, thus, results of Section 4.1 can be directly applied.

All weight vectors with magnitude $|\mathbf{W}|$ identify a hypersphere of radius $|\mathbf{W}|$ centered in the origin. Distribution of weight vectors over such hypersphere is assumed uniform. The ratio between the number of weight vectors inducing output errors and the total number of weight vectors on a hypersphere is identical for every hypersphere. Therefore, $P_i(\mathbf{X}_e)$ can be computed by referring to a generic $\mathbf{W}$. Rotation θ of the input vector 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 $\mathbf{X}$ as an error $\mathbf{W}_e$ on $\mathbf{W}$ by making the appropriate substitutions. Equivalence of these views is granted when the following equation holds:

$$\mathbf{W}_e \cdot \mathbf{X} = \mathbf{W} \cdot \mathbf{X}_e \quad (8)$$

To identify the weight-equivalent error, we are looking for a solution $\mathbf{W}_e$ of equation (8), where $\mathbf{X}$, $\mathbf{W}$, and $\mathbf{X}_e$ are known values. By subtracting $\mathbf{W} \cdot \mathbf{X}$ from both

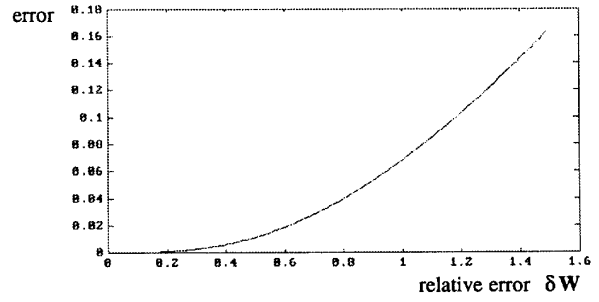


Fig. 6 - The approximation error of Widrow's approach with respect to our approach

terms, we obtain:

$$|\mathbf{W}_e - \mathbf{W}| |\mathbf{X}| \cdot \cos \mu = |\mathbf{W}| |\mathbf{X}_e - \mathbf{X}| \cdot \cos \nu \quad (8')$$

where μ is the angle between $\Delta \mathbf{W}$ and $\mathbf{X}$, and ν is the angle between $\mathbf{W}$ and $\Delta \mathbf{X}$. When $|\mathbf{W}|$ and $|\mathbf{X}|$ are not null, this leads to:

$$\begin{aligned} \delta W &= \frac{|\mathbf{W}_e - \mathbf{W}|}{|\mathbf{W}|} = \\ &= \frac{|\mathbf{X}_e - \mathbf{X}|}{|\mathbf{X}|} \cdot \frac{\cos \nu}{\cos \mu} = \delta X \cdot \frac{\cos \nu}{\cos \mu} \end{aligned} \quad (9)$$

Equation (9) has an infinite number of solutions, being $|\mathbf{W}|$ defined over the complete real set. In particular, we can safely assume $\cos \nu = \cos \mu$, by which we finally obtain $\delta W = \delta X$. By applying the same statistical analysis performed in Section 4.1 and by replacing the expression of δX , we derive for the case of a single-step function:

$$P_x(\delta X) = \frac{1}{2\pi} \left[\frac{\pi}{2} + 2 \cdot \text{atg} \left(\frac{\delta X - 1}{\delta X + 1} \right) \right] \quad (10)$$

It may be noticed that such result holds for all possible values of δX , i.e., without the restrictions introduced in [7]. Besides, for small values of δX , equation (10) provides the same results given in [7].

4.3 The Neuron's Error Probability for Synaptic Product Errors

As for errors affecting a synaptic product, treatment can be reduced to that of weight errors (see section 4.1) provided a formal correspondence between the product error and an "equivalent" weight error is defined. To this end, let p_z be the affected product; the product error ratio δp_z is given by $\delta p_z =$

$|\Delta p_z|/|p_z|$. The equivalent weight error ΔW , which induces an unexpected value in the σ_e in the summation, can be derived as follows:

$$\begin{aligned}\sigma_e &= \sum_i W_i X_i + \Delta p_z = \\ &= \sum_{i \neq z} W_i X_i + W_z X_z + \Delta p_z = \sum_{i \neq z} W_i X_i + W_z^* X_z\end{aligned}\quad (11)$$

From the above notation, it follows:

$$\begin{aligned}\delta W &= \frac{|\Delta p_z|}{|X_z|} \cdot \frac{1}{|\mathbf{W}|} = \\ &= \delta p_z \cdot \frac{|p_z|}{|X_z| \cdot |\mathbf{W}|} = \delta p_z \cdot \frac{|W_z|}{|\mathbf{W}|}\end{aligned}\quad (12)$$

4.4 The Neuron's Error Probability for Summation Errors

In the case of errors affecting the input summation, the probability $P_\sigma(\Delta\sigma)$ of observing an error $\Delta\sigma$ at the neuron output is given by the ratio between the number of misclassified input vectors and the total number of input vectors. Evaluation of such a probability may be performed by following the same guidelines described in Section 4.1 for weight errors and by adopting the geometrical interpretation discussed in Section 3.4. A summation error $\Delta\sigma$ shifts the hyperplanes by a segment $\Delta\sigma$. The misclassified input vectors belong to the volume of the hyperspheric segment limited by the correct and the shifted hyperplanes on the hypersphere. $P_\sigma(\Delta\sigma)$ 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 the probability $P_\sigma(\delta\sigma)$ of observing the relative error $\delta\sigma$ at the neuron's output. Still, the above computation is far from simple.

In the case of a single-step evaluation function, the same results can be achieved by the following remarks. Given any $\sigma = \bar{\sigma}$, the probability $P_\sigma(\delta\sigma, \bar{\sigma})$ of observing the relative error $\delta\sigma$ is:

$$P_\sigma(\delta\sigma, \bar{\sigma}) = \begin{cases} 0 & \text{if } \delta\sigma \leq 1, \text{ i.e., } |\Delta\sigma| \leq |\bar{\sigma}|, \\ \frac{1}{2} & \text{if } \delta\sigma > 1, \text{ i.e., } |\Delta\sigma| > |\bar{\sigma}|. \end{cases}\quad (13)$$

This result is intuitively justified by observing that if $|\Delta\sigma| \leq |\bar{\sigma}|$ the result of the evaluation function is the correct one; otherwise, given a uniform distribution of the error values, the probability of obtaining an unexpected result at the neuron's output is 0.5.

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

$$\begin{aligned}P_\sigma(\delta\sigma) &= \int_{-\infty}^{+\infty} P_\sigma(\delta\sigma, \zeta) p(\zeta) d\zeta = \\ &= \begin{cases} 0 & \text{if } \delta\sigma \leq 1, \\ \frac{1}{2} \int_{-\infty}^{+\infty} p(\zeta) d\zeta = \frac{1}{2} & \text{if } \delta\sigma > 1. \end{cases}\end{aligned}\quad (14)$$

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

4.5 The Neuron's Error Probability for Activation Function's Errors

Probabilistic analysis of activation function's errors is very 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. In other words, 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.

4.6 Evaluation of 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 **E** affects the whole computation.

Any single error affecting (as seen in the previous sections) a neuron in layer f ($1 \leq f < L$) can be seen as creating an error on one input to 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 the computation is iterated 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 ($f+1 \leq l \leq L$). 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 $\mathbf{X}_l$ into the layer's output vector $\mathbf{O}_l$ and the (possible) input error $\Delta \mathbf{X}_l$ into the output error $\Delta \mathbf{O}_l$. Let δO_l be the output error ratio ($\delta O_l = |\Delta \mathbf{O}_l| / |\mathbf{O}_l|$). 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:

$$p_l(\delta O_l) = \sum_{\delta X_l} p_{\delta O_l | \delta X_l} \cdot p(\delta X_l) \quad (15)$$

where the summation over all possible values of δX_l is substituted by an integral when δX_l has continuous values.

Consider as a working example neurons characterised by a single-step activation function. For any value δX , the value of δO is either 0 or 2; in fact, the output error $\Delta \mathbf{O}$ of a single-step activation function - if any - is either +2 or -2, while $|\mathbf{O}| = 1$. By applying the procedure in Section 4.2, we evaluate the conditional probability of observing the error related to each particular value of δX . This function is:

$$p_{\delta O_l | \delta X_l} = P_x(\delta X_l) \cdot \delta(2) \quad (16)$$

where $\delta(2)$ is the Dirac's function centered in $\delta O = 2$. Therefore, the total probability density function $p(\delta O_l)$ will be determined as the weighted summation of such conditional probabilities; in our case, it is:

$$p(\delta O_l) = \delta(2) \cdot \sum_{\delta X} P_x(\delta X_l) \cdot p(\delta X_l) \quad (17)$$

By iterating the above procedure, we obtain the probability density functions for each 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 $P^*(\delta E)$.

5 Concluding Remarks

Equation derived in this paper can be used as guidelines during design of artificial neural networks. To provide an example of such 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 [19], 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 micron CMOS technology; 8 bits have been chosen for weight representation and 13 bits 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 1 has been derived: for different classes of defects, we evaluated the total number of defect 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 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 output. As a consequence, fault-tolerance policies (or, at least, concurrent fault-detection techniques) should be concentrated in these two subsystems. It appears then that coding solution such as the one presented in [20], adopting arithmetic

Defect Type	Number of Defects in the Network	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	0.0334	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

Table 1 - A sample evaluation of the error probability in the presence of production-time defects

codes, would be suitable. In the reference here recalled, AN codes are used as granting an acceptable compromise between detection capability and redundancy.

References

- [1] C. Neti, M.H. Schneider, 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, W. Maly: "Behavioral modeling of physical defects in VLSI neural Networks", *Proc. 1990 Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Grenoble, France, Nov. 1990
- [3] D.B.I. Feltham, W. Maly: "Limitation to the size of single-chip electronic neural networks", *Proc. IEEE International Conference on WSI*, San Francisco, CA, Jan. 1991
- [4] F. Distanti, M. Sami, R. Stefanelli, G. Storti-Gajani: "Mapping neural nets onto a massively parallel architecture: a defect-tolerance solution", *IEEE Proceedings*, Vol. 79, No. 4, pp. 444-460, Apr. 1991
- [5] V. Piuri, M. Sami, R. Stefanelli: "Fault tolerance in neural networks: theoretical analysis and simulation results", *Proc. Compeuro 1991*, Bologna, Italy, May 1991
- [6] L.A. Belfore II, B.W. Johnson: "The Analysis of the Faulty Behaviour of Synchronous Neural Networks", *IEEE Transaction on Computers*, Vol. 49, No. 12, pp. 1424-1429, Dec. 1991
- [7] M. Stevenson, R. Winter, B. Widrow: "Sensitivity of feedforward neural networks to weight errors", *IEEE Trans. Neural Networks*, Vol. 1, No. 1, pp. 81-92, March 1990
- [8] C. Alippi: "Asymptotic insensitivity to weights perturbations in back-propagation classifiers", *Proc. IJCNN93*, Nagoya, Japan, Oct. 1993
- [9] S. Piche: "The effects of Weight errors in Neural Networks", *Proc. WCNN93*, Portland, OR, July 1993
- [10] C. Alippi: "Sensitivity of Artificial Neurons to weights and input quantization: 1. the input neurons case", *Proc. WCNN93*, Portland, OR, July 1993
- [11] C. Alippi: "Sensitivity of Artificial Neurons to weights and input quantization: 2. the hidden neurons case", *Proc. WCNN93*, Portland, OR, July 1993
- [12] J. Durham, E. VonColln: "A measure of Signal to Noise ratio for multilayer perceptrons trained as 0-1 classifiers", *Proc. WCNN93*, Portland, OR, July 1993
- [13] C. Alippi, L. Briozzo: "Setting and validating precision requirements in the digital VLSI implementation of a neural defect-identifier for mechanical objects", *Proc. ISCAS'94*, London, UK, June 1993

- [14] C. Alippi, V. Piuri, M.G. Sami: "Sensitivity to errors in artificial neural networks: a behavioral approach", Report No. 089.93, Dept. of Electronics and Information, Politecnico di Milano, Milano, Italy, 1993
- [15] J. Hertz, A. Krogh, R.G. Palmer: *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Co., 1991
- [16] M.E. Hoff, Jr.: *Learning phenomena in networks of adaptive switching circuits*, Ph.D. dissertation, Dept. Electrical eng., Stanford Univ., Stanford, CA, June 1962
- [17] F.H. Glanz: *Statistical extrapolation in certain adaptive pattern recognition systems*, Ph.D. dissertation, Dept. Electrical Eng., Stanford Univ., Stanford, CA, May 1965
- [18] 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
- [19] C. Lehmann, F. Blayo: "A VLSI implementation of a generic systolic synaptic building block for neural networks", Proc. Int'l Workshop on VLSI for artificial intelligence and neural networks, Oxford, UK, 1990
- [20] V. Piuri, M. Sami, R. Stefanelli: "Arithmetic codes for concurrent error detection in artificial neural networks: the case of AN+B codes", *Proc. 1992 Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Dallas, TX, Nov. 1992