

AN ALGORITHMIC APPROACH TO CONCURRENT ERROR DETECTION IN ARTIFICIAL NEURAL NETWORKS

Vincenzo Piuri

Department of Electronics, Politecnico di Milano
piazza L. da Vinci 32, I-20133 Milano, Italy

Abstract - Fault tolerance is a basic issue for VLSI implementation of artificial neural networks dedicated to mission-critical applications. In this paper, we propose a high-level approach to concurrent error detection: our technique is based upon the behavioral definition of the neural computation and abstracts from the specific architectural and technological implementation.

INTRODUCTION

Artificial Neural Networks (ANNs) are nowadays increasingly studied and used in a variety of applications, when no algorithmic approach is known efficient. A number of researches were performed and are in progress for pattern recognition, for classification, for signal and image processing [1] [2]. Wide diffusion of architectures implementing such computational paradigms needs fault-tolerance features, which allow error detection, error correction and system reconfiguration. Continuous operation and mission critical applications often require a concurrent approach to fault tolerance, i.e., identification and treatment of errors concurrently with the nominal computation.

Some researches afforded the fault-tolerance issue by analyzing the characteristics of hardware faults [3] and the influence of faults onto the neural computation and system survival [4]. Error detection and, possibly, correction have been also considered from the point of view of the specific hardware implementation [2]; the proposed solutions were designed and superimposed on the basic circuits, without considering the computation performed by such circuits: they are in fact an adaptation of general fault-tolerance techniques for VLSI circuits to neural circuits. Hardware reconfiguration has been considered by different researchers for highly-regular implementations [5] [6] [7]. At the moment, no consideration has been given to the definition of the fault-tolerance characteristics at behavioral level in the neural computation.

Different approaches can be envisioned to improve the neural computation in order to support concurrent fault tolerance:

- *Data coding* substitutes each input pattern with a coded representation; the neural computation is performed upon the coded information; the final output is decoded and checked to detect the error occurrence [8].
- *Intrinsic fault tolerance* improves the computational structure (i.e., the number of neurons and interconnections) and redistributes the neural

activities on the enlarged network. It has been proved that high error masking and system survival can be achieved [4], despite a reduction in generalization capabilities.

- *Algorithmic redundancy* modifies the neural computation by performing a redundant application-specific computation in parallel to the nominal one; information generated by the redundant computation are used to check the correctness of the nominal results.

The first two techniques are general approaches, which do not fully exploit the characteristics of the neural computation. This implies that a large amount of redundancy may be required and, in turn, a large number of circuits must be added to the basic implementation.

In this paper we propose the use of algorithmic redundancy to introduce concurrent error detection capabilities and, possibly, error correction. After a brief summary of neural computation and the behavioral error model, the methodology for algorithmic redundancy is discussed and evaluated with reference to some applications.

THE NEURAL COMPUTATION AND THE ERROR MODEL

Artificial neural networks are computational models capable to exploit massive computational parallelism [1] [2] [9]. This characteristic is due to the holographic distribution of computation and information in the whole computing system. Typical applications are a number of problems for which no effective procedural solution is known, e.g. classification problems of uncertain, or incomplete, or noisy patterns, and multiple-goals optimizations.

ANNs contain a high number of computing elements (the *neurons*), which are (often densely) connected by weighted interconnections (*synapses*). The output of each neuron is generated by applying a non-linear function to the weighted sum the neuron's inputs. Neurons are often grouped into homogeneous clusters (*layers*).

A number of specialized neural models were proposed in literature to solve specific classes of applications [1] [2] [9]. The model of the adopted neural computation may be tailored upon the specific application by using an incremental procedure based on *learning* of the nominal behavior. Sequences of examples are supplied during training: knowledge is acquired and stored in the network by properly updating the interconnection weights. The configuration is then frozen and the *recall* phase of the ANN is executed on the basis of the learnt behavior. Different learning procedures were proposed and evaluated in literature. In *supervised* learning techniques, the learning phase adjusts the synaptic weights to minimize the difference between the actual outputs and the expected ones; *unsupervised* learning techniques update the synaptic weights by extracting information only from the nominal inputs, without any knowledge about the expected outputs.

In our research we considered one of the main classes of neural models, namely the multi-layered feed-forward networks [9], since this class has found many applications in the field of signal and image processing, e.g., for filtering and

pattern classification. However, our technique can be extended to deal also with other kinds of neural paradigms.

In order to describe the erroneous computation, due to fault occurrence in the architecture implementing the ANN, we define a behavioral error model of the neural paradigm, independent from the technological implementation. Errors here taken into account belong to any of the subsequent classes [4]:

1. *synaptic errors* produce an unexpected value of the weight-signal product, due to an error affecting either the synaptic weight or the associated multiplication;
2. errors affecting the *summation* of the weighted inputs;
3. errors affecting the *non-linear evaluation function*.

Besides, we must consider also errors due to faults in devices related to system-level activities (e.g., power supply, weight-value refresh circuit): they are more critical since they often affect in a fatal way the operation of the whole system.

Mapping of faults onto errors is strictly related to the specific implementation. When direct mapping of the neural computation is performed onto the hardware implementation (e.g., as in analog solutions or in some digital structures [7]), correspondence between faults and errors is straightforward. Whenever digital architectures adopt time-multiplexing of some devices (e.g., weight multipliers), single faults may appear as multiple neural errors. In our research, we considered all the above cases: to evaluate the effectiveness of our approach, we performed a number of simulations for some representative architectures [6] [10] [11] by adopting a random mixture of single and multiple errors.

THE ALGORITHMIC APPROACH TO FAULT TOLERANCE

To introduce fault-tolerance characteristics in the neural implementation we explored a high-level approach based upon the behavioral definition of the neural computation. Algorithmic redundancy abstracts completely from the specific architecture, implementation or technology adopted to realize the circuit for the considered neural computation.

The basic idea of algorithmic redundancy consists of the identification of an upgraded neural algorithm: the basic part of such computation generates the nominal results as required by the application, while the additional computation provides the redundant information for error detection and, possibly, correction. Algorithmic redundancy does not use data coding at the level of each single data to execute the redundant computation; it may be viewed as coding upon subsets of input data. This technique was firstly proposed in literature for concurrent error detection in traditional architectures, e.g., in array processors for convolution [12].

According to the specific nominal algorithm, additional input data are generated from subsets of the nominal inputs by applying simple mathematical operators; such additional data are treated as in the nominal computation by the same network, or by a redundant one (i.e., by an architecture with

a number of neurons and interconnection paths greater than the minimum required to solve the specific problem), or by a completely separated network. Then, checking information is extracted from the redundant outputs by means of simple mathematical relationships: such information is used to test the correctness of the nominal outputs.

The rules for generating the additional inputs and for extracting the checking information are strictly dependent on the applications considered, i.e. on the nominal neural algorithm, and have to be identified for each specific case by inspecting the mathematical characteristics of the computation. The algorithmic approach exploits in fact specific relationships which hold between subsets of the nominal input data and which are preserved (or modified in a known way) by the nominal neural computation. However, the redundant computation performed upon additional data is usually very similar to those executed upon the nominal data; often, it is the same kind of computation.

Two basic approaches may be considered for generating and treating redundant data, according with the kind of mathematical relationships that have been identified among the nominal data: *combined data* and *grouped data*.

In the case of *combined data*, the additional data are obtained by applying a suitable combination rule (e.g., a linear combination) to the nominal data coming from the external environment. The additional data are therefore interleaved with the nominal ones: after presentation of a set of nominal input data at the primary inputs of the ANN, the additional data created from such set of nominal data are supplied to the network itself. Computation upon redundant data is identical to the nominal one; thus, it is performed by the same network of the nominal one and no additional training is required. In Fig. 1.a we give the overall scheme of the architecture for this approach. A very low hardware redundancy is required: neither additional neurons nor interconnection paths must be added to the nominal network, but only the circuits for computing the additional input data and for checking the final outputs. On the other hand, interleaving of the nominal and the redundant computations increases the computation time and introduces periodic testing (i.e., reduces the overall availability of the ANN).

The *grouped data* are useful, for example, in classification problems. In these cases, the input data are naturally classified in sets by the nominal computation. The redundant computation introduces a different repartition of data into classes (possibly, disjoint) according to specific rules, related both to the specific application, and to the distribution of input data, and to the requirements for fault tolerance. The redundant data are therefore identical to the nominal ones; for each nominal data, the redundant network performs the redundant computation, i.e., it performs (on the nominal inputs) operations which are similar to the nominal operations. In Fig. 1.b the overall architecture for this approach is given. Hardware redundancy consists of neurons for redundant computation and of checking circuits; computational time is not increased since checking is performed in parallel to the nominal computation. For the redundant computation, it is usually necessary to complete the training of the redundant ANN to tailor the uncommitted parameters of

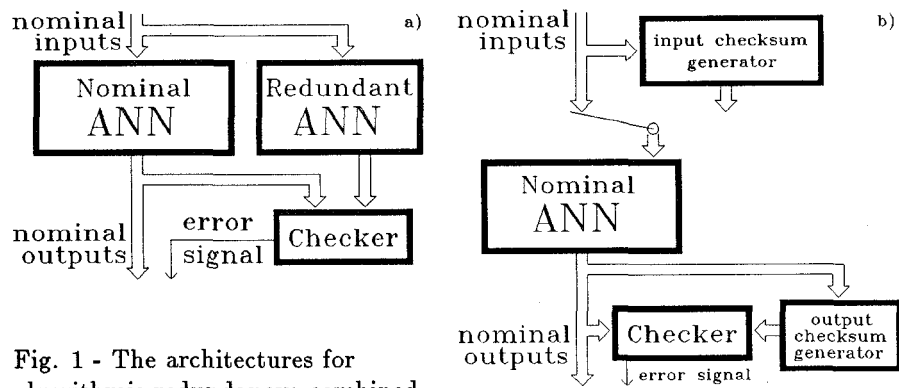


Fig. 1 - The architectures for algorithmic redundancy: combined data (a) and grouped data (b).

the redundant architecture and to introduce the fault-tolerance capabilities in the whole structure; the redundant network behaves as any other ANN. In few cases, the redundant network can be designed *a priori* by exploiting the characteristics of the application and the tailored parameters of the nominal ANN. For example, in classification problems with linearly-separable classes, the input classes may be grouped and the interconnection weights may be extracted from nominal tessellation of the input space to generate a redundant tessellation. for the redundant computation.

Since the algorithmic redundancy approach to concurrent fault tolerance is strictly related to the specific application, the use of this technique must be carefully considered for each specific case. In this paper, we give the general framework and the methodology for this approach. Evaluation of effectiveness and efficiency is also discussed for two examples related to signal and image processing. Algorithmic redundancy can be considered attractive for a specific application only if the overall costs (i.e., the circuit complexity and the increase of computation time) are reasonably small in comparison with the fault-tolerance capabilities (namely, error detection and correction); as reference solution we may consider modular redundancy or other architectural- and implementation- specific techniques.

CASE STUDIES: APPLICATION AND EVALUATION

In this section we discuss the use of algorithmic redundancy for concurrent fault tolerance in two specific applications, which are particularly important for signal and image processing: filtering/convolution and classification. A detailed evaluation is given by presenting results of extensive simulations performed with a number of different input distributions and fault distributions.

Filtering and convolution

In FIR filtering and convolution, we can compute an algorithm-related checksum for a sequence of network inputs. The continuous flow of nominal inputs may be separated into finite-length sequences: the checksum of each

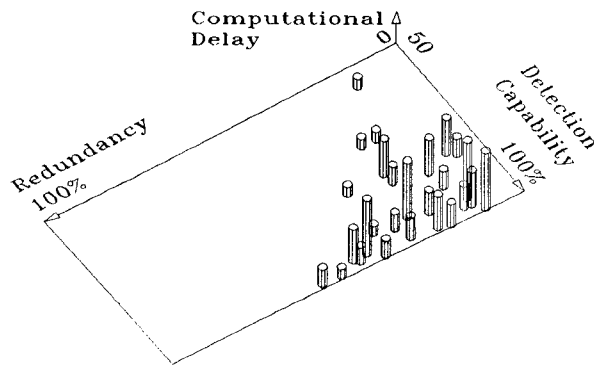


Fig. 2 - Redundancy, detection probability and computational delay

sequence may be obtained by adding the inputs of the sequence. It has been proved [12] that filtering applied to the checksum may give information about correctness of filtering applied to the sequence of data. The nominal computation is executed upon the nominal data and, then, upon the redundant data. Nominal outputs are added together as soon as they are produced, in order to generate the output checksum. An error is detected whenever the result obtained from the redundant input differs from the output checksum. The redundant hardware is composed by the addition stage at the primary inputs to generate the redundant input, by the addition stage to compute the output checksum and the final comparator to detect the presence of the possible error. The computational time is increased with respect to the nominal computation since, at each new input, two addition must be computed (one before the input layer, the second one after the output layer). Besides, at the end of each sequence of input data a complete computational step of the network is dedicated to generate the output checksum (i.e., it is lost from the point of view of the nominal computation).

Some results of the experimental use of this approach to feed-forward multi-layered networks and to Hopfield's networks for filtering applications are shown in Fig. 2; this is not an exhaustive evaluation of the capabilities of the approach since effectiveness is strictly related to the application, but it shows the overall capabilities that can be achieved. For our experiments, we considered a number of input distributions, different architectural implementations, and a number of error distributions. We evaluated the hardware redundancy (i.e., the number of additional components that must be added to the nominal network), the detection probability (i.e., the probability that the injected error is detected by the redundant network), and the computational delay (i.e., the additional time required to generate the final outputs).

Pattern classification

In classification problems, the input patterns are divided into a set of classes. During recall, the ANN's operations lead to identify the class to which every input pattern belongs. These classes of the input space can be grouped in super-sets or, more in general, the input patterns can be reclassified into super-sets according to a suitable criterion. This operation

reduces the number of classes that must be recognized and the size of the redundant ANN with respect to the nominal network.

Each nominal input data is treated by the nominal ANN, which performs the nominal computation: the output identifies the class to which the input belongs; in parallel, the redundant network identifies the super-sets to which the input belongs. Check on the final output means to verify that the class, identified by the nominal classification, belongs to the super-set recognized by the redundant ANN.

Grouping of the input classes is obviously not unique. A number of criteria may be adopted to define the super-sets by taking into account the characteristics of the input classes (e.g., linear separability), the size of the resulting network for super-set classification, and the error detection capabilities. Two main approaches have been explored: *adjacent fusion* and *re-classification*.

In *adjacent fusion*, we collapse two or more adjacent classes into a single super-set by removing the boundaries which separate the classes themselves. For example, in linearly-separable classes, each boundary is associated to a neuron in the first layer. Cancellation of a boundary implies cancellation of a neuron in the first layer; this, in turn, forces the possible other pairs of classes separated by such boundary to collapse into new super-sets. An example is shown in Fig. 3.a. The choice of the boundaries that must be removed (i.e., the low-granularity tessellation in super-sets) is not unique: some guidelines have been shown effective by extensive experiments but no general rule may be identified due to the high dependency on the specific tessellation and on the specific distribution of actual input data in the considered tessellation. The following guidelines for boundary cancellation have been successfully experimented:

- in two adjacent classes, if the probability that a fault leads to classify a pattern belonging to one class as a pattern of the other class is reasonably low, the boundary can be removed;
- if patterns of two adjacent classes have high probability of occurring, we should not remove the boundary since the probability of error masking (i.e., the probability that a pattern belonging to one of such classes is assigned to the other one due to a fault) may become non-neglectable; on the contrary, if the probability is very small, it is reasonable to remove the boundary;
- if the actual input patterns belonging to two adjacent classes are enough far from the common boundary, we can remove the boundary itself since a reasonably-small change of the boundary position usually does not affect the classification;
- if the meaning of two adjacent classes or the actions associated to them are similar, the boundary can be removed.

Other criteria can be deduced from the specific application.

As the redundant network is concerned, it may range from a very small network with poor classification granularity to a complete duplication of the nominal network according with the number of boundaries that have been removed. No computational time is added to the nominal time. The nominal

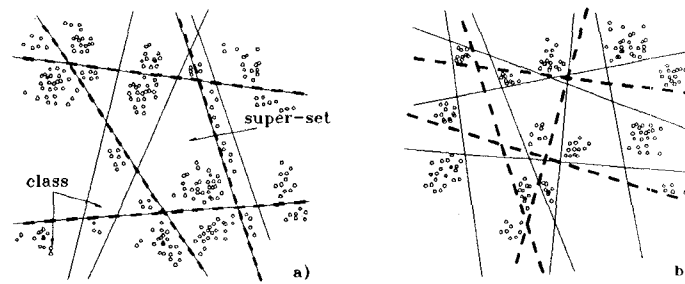


Fig. 3 - Classes and super-sets: adjacent fusion (a), re-classification (b)

outputs are delivered at the same speed and throughput of the nominal architecture. Such speed and throughput constitute also the upper bound for the performances of the redundant network; possible speed increase may be due to the reduced number of operands in the input summation in each neuron. Checking and validation of the final outputs of the network are performed after the generation of such outputs since nominal and redundant outputs are available at the same time; these operations may be executed in parallel with the computation of the subsequent outputs since time for checking can be practically neglected with respect to the nominal computation time. Otherwise, if checking and validation must be performed before starting the next computational cycle upon a new input pattern, the checking time must be added to the nominal computation time of the ANN.

In the case of a very small number of super-sets with respect to the number of classes, the hardware redundancy is very low, but, unfortunately, the error detection is generally very poor since few redundant information is stored into the network. At the opposite limit, the complete duplication of the network impose a massive hardware redundancy, but guarantee the maximum capability of detecting all errors which are not masked by the characteristics of the neural computation itself (due to non-linearities of the evaluation functions and to reconvergent fan-out). For example, error masking occurs whenever a pattern belonging to a given class is classified in another class in the same super-set of the first one due to a fault in the network performing the nominal computation. A reasonable balancing of the previous conflicting requirements must be carefully considered on the specific application.

In *re-classification*, a new distribution of the input space in super-sets is performed by introducing a reduced number of super-sets with respect to the number of classes and by adopting boundaries between super-sets that do not necessarily coincide with the boundaries between classes. In other words, a new classification problem is considered on the same input space of the nominal algorithm. An example is shown in Fig. 3.b. This approach is often useful for classes which are not linearly-separable with complex boundaries. In fact it allows to reduce drastically the number of neurons required for checking, even if detection capabilities may be smaller than in the adjacent fusion. Also computation time and delay due to checking are identical to those discussed above for such approach.

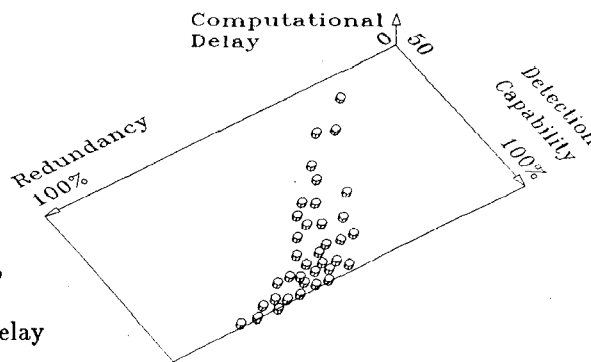


Fig. 4 - Redundancy, detection probability and computational delay

Even if a quantitative absolute evaluation of the performance of algorithmic redundancy for pattern classification is meaningless since it is strongly dependent on the application, we present some results of our experiments in Fig. 4 to give at least an idea of the effectiveness of the approach. For our experiments in pattern classification we considered a number of input distributions, a number of classes, the different techniques for creating the redundant computation, different architectural implementations, and a number of error distributions. For each experiment we optimized the redundant network and we evaluated the hardware redundancy (i.e., the number of additional components that must be added to the nominal network), the detection probability (i.e., the probability that the injected error is detected by the redundant network), and the computational delay (i.e., the additional time required to generate the final outputs).

Error correction needs additional redundancy to re-generate the correct output even in presence of errors in the output of the nominal network. A feasible approach with algorithmic redundancy consists of adding a second redundant network capable (together with the redundant network for error detection) of supplying enough information in order to mask the error.

A first solution consists in duplicating the nominal network: the first nominal network perform the nominal computation, the redundant network for error detection validates the output generated by such nominal network, and the second nominal network provides the correct results whenever the first one is faulty. Obviously, the checking circuit must be improved to detect which network is faulty by inspecting the outputs produced by the two nominal networks and by the one for error detection.

A second approach is based upon introduction of a second redundant network designed as discussed for the one related to error detection. In designing the second redundant network we must consider the following constraint: the tessellation of the nominal network must be equal (or at least very similar) to the tessellation defined by considering, at the same time, the boundaries of the super-sets recognized by each of the redundant networks. This constraint requires that a class is the intersection between only one super-set of the first redundant network and only one super-set of the second one. The nominal network computes the nominal outputs, while both the first and the

second redundant networks validate these outputs. For single errors, in the case of erroneous outputs detected by both of the redundant networks, the correct output is the class corresponding to the intersection of the super-sets identified by such networks. Correction is much more difficult when only one redundant network detects an error in the nominal computation. If the super-sets identified by the redundant networks are disjoint, the error is in the redundant network which signaled the error: no output correction is required. Otherwise, the correct output belongs to the intersection between the super-sets recognized by the redundant networks.

By adopting a fault-tolerant checker, in both the previous cases, a high value of error correction (i.e., the percentage of correct outputs even in presence of faults) is achieved for single errors; when the assumption of single errors in the neural computation is reasonable, about 99.6 % has been reached in the first case, while about 98 % in the second one.

Further researches will investigate the problem of foreseeing the redundancy required by a specific application to achieve given fault-tolerance capabilities.

REFERENCES

- [1] J.A. Anderson, E. Rosenfeld, editors, *Neurocomputing: foundations of research*, The MIT Press, 1988
- [2] *Proc. IJCNN 92*, Baltimore, USA, 1992
- [3] D.B.I. Feltham, W. Maly, "Behavioral modeling of physical defects in VLSI neural networks", *IEEE Proc. Conf. on WSI*, San Francisco, 1991
- [4] V. Piuri, M. Sami, R. Stefanelli, "Fault tolerance in neural networks: theoretical analysis and simulation results", *IEEE Proc. Compeuro 91*, Bologna, Italy, 1991
- [5] F. Distanto, et al., "Fault tolerant characteristics of the linear array architecture for WSI implementation of neural nets", *IEEE Proc. Conf. on WSI*, San Francisco, 1991
- [6] 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
- [7] D. Zhang, G.A. Jullien, W.C. Miller, E. Swartzlander: "Arithmetic for Digital Neural Networks", *IEEE Proc. ARITH10*, Grenoble, 1991
- [8] V. Piuri, R. Stefanelli, M. Sami, "Concurrent Error Detection in Artificial Neural Networks: the Use of AN+B Codes", *IEEE Proc. 1992 Defect and Fault Tolerance in VLSI Systems*, Dallas, 1992
- [9] J.Hertz, A.Krog and R.G. Palmer: "Introduction to the Theory of Neural Computation", Addison-Wesley, 1991
- [10] C. Mead, *Analog VLSI and neural systems*, Addison-Wesley, 1989
- [11] S.Y. Kung, "Parallel architectures for artificial neural nets", *Proc. Systolic Arrays 1988*, San Diego, USA, 1988
- [12] J.A. Abraham, P. Banerjee, C. Chen, W.K. Fuchs, S. Kuo, A.L.N. Reddy, "Fault-tolerance techniques for systolic arrays", *IEEE Computer Magazine*, July 1987