ARITHMETIC CODES FOR CONCURRENT ERROR DETECTION IN ARTIFICIAL NEURAL NETWORKS: THE CASE OF AN+B CODES

Vincenzo PIURI, Mariagiovanna SAMI, Renato STEFANELLI

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

Abstract

A number of digital implementations of neural networks have been presented in recent literature. Moreover, several authors have dealt with the problem of fault tolerance; whether such aim is achieved by techniques typical of the neural computation (e.g., by repeated learning) or by architecture-specific solutions, the first basic step consists clearly in diagnosing the faulty elements. In the present paper, we suggest adoption of concurrent error detection; the granularity chosen to identify faults is that of the neuron. An approach based on a class of arithmetic codes is suggested; various different solutions are discussed, and their relative performances and costs are evaluated. To check the validity of the approach, its application is examined with reference to multi-layered feed-forward networks.

1. Introduction

Artificial Neural Networks (ANNs) offer an attractive solution approach for the increasing demand of massive computation in many critical application areas (e.g. signal and image processing, real-time control, etc.). Advances in integration technologies and architectural design now allow implementations at reasonable costs: even commercial systems are by now available to large classes of users.

The possibility of VLSI or WSI implementation of ANNs and their application in mission-critical areas lead to consider the associated problems of defect and fault tolerance. Several authors have dealt up to now with such problems in relation with specific implementations of neural nets. Thus, for example, in [1] behavioral fault models are defined with reference to multi-layered nets implemented by analog systems, and the impact of the physical faults is examined at various abstraction levels. Following the same line, in [2] a relationship is identified between physical defects and failures and the maximum size of the device, again corresponding to a given analog implementation approach. Other studies evaluate the intrinsic robustness of the neural paradigm [3] [4]; this implies the definition of a behavioral error model at a very high abstraction level (related to the abstract neural model) and the evaluation of such errors' effects onto the neural computation, in a technology-independent way. In [5] redistribution of the computation and information has been discussed to minimize the influence of faults onto the computation itself by exploiting the intrinsic fault tolerance of the network.

If, on the contrary, an implementation-related approach is taken, architectural solutions must be taken into account. A preliminary step consists in the identification of the faulty element present in the network and in its subsequent confinement. External testing – performed off-line – can detect the presence of a fault, but exact identification of the faulty element is much more difficult. Conditions granting behavioral testability for multi-layered feed-forward networks are presented in [6]. In the present paper, we suggest instead to use concurrent-detection techniques suitable for *digital* implementations of neural networks; while the solutions outlined are developed for the individual neuron and as such could be extended to arbitrary networks without impacting on the network's topology, we will analyze performances and costs for the particular case of multi-layered feed-forward networks. Concurrent techniques, using the nominal data to perform error detection, are particularly suited to mission-critical systems, where high system credibility is the first mandatory requirement.

We consider first a very simple model for the individual neuron, derived from its basic functionality as defined by $x_i = f_i(\sigma_i - \theta_i) = f_i(\sum_j w_{ij}x_j - \theta_i)$, where x_i is the output signal of the neuron, $\sigma_i = \sum_j w_{ij}x_j$ is the sum of the weighted inputs, θ_i is a threshold value and f_i is an (a-priori, arbitrary) non-linear evaluation function.

The simplest, immediate implementation of such equation involves one-to-one mapping of the equation operators onto digital components; weights are stored in memory elements, products $w_{ij}x_j$ are evaluated by digital multipliers, the summation is performed by an n-input adder and a suitable circuit implementing the evaluation function generates the neuron's output signal. The fault model we adopt to define our fault-detection technique is based upon this structure and it locates the possible faults within the components of the architecture (weight storage, arithmetic devices, function evaluator); as usual in most fault-tolerance approaches, interconnections are assumed fault-free, unless otherwise specified. Even though we do not for the moment make any specific assumption on the technological implementation of such devices, we restrict our analysis to systems adopting *fixed-point arithmetics* (with the related implications on the functional error model).

Given such model of the neuron, first of all we will discuss different modified neuron architectures, all based on the adoption of a class of arithmetic codes. The basic concepts of such codes will be summarized in section 2, while the different neuron structures and the sets of errors for which each grants detection will be analyzed in section 3. In section 4, a comparative evaluation of the different solutions (assuming one-to-one mapping of the neural network onto a corresponding digital architecture) will be discussed.

2. The class of arithmetic codes adopted: $AN + B$ codes

Having chosen to make the individual neuron's structure capable of concurrent fault detection, several architectural possibilities arise to achieve this goal. Focussing on arithmetic codes is justified by a number of reasons: while being cost-effective as far as structure redundancy is concerned, they grant efficient detection without any relevant time overhead. Moreover - and most important - they will be seen to support effectively the fault models we are concerned with, not only in the case of the arithmetic units present in the neuron's structure but also for the other components (in particular, storage units and function evaluator). Before discussing the modified neuron's structure, we will briefly review the class of codes adopted and their characteristics.

The main requirements guiding in choice of an arithmetic code for our application can be summarized as follows:

a. error detection only is requested; the single-error assumption is accepted;
b. given the great complexity of neural networks envisioned for real-life applications, it is important to keep the silicon overhead for the single neuron as low as possible;
c. the neural computation must not be modified; this is essential to grant that capabilities of the network (learning, recall, generalization) be not affected with respect to the initial, theoretical definition;
d. the "hard-core" section, i.e., the subsystem not capable of error detection (and whose faults would therefore be fatal to system's credibility) must be as small as possible: this implies, in particular, design of "fail-safe" decoders.

We identified $AN + B$ codes as capable of satisfying all the above, given reasonable choices for A and B [7].

AN codes (the simpler subset of $AN + B$ codes) are non-systematic codes in which each nominal datum N is substituted by its coded representation $C(N)$ via the linear transformation $C(N) = A \times N$, where the integer constant A is the ***code generator***. Arithmetic units – adders, subtractors, multipliers – are not modified by the adoption of the coded data; the only difference with respect to the nominal structure is given by the number of bits required, that obviously increases with A. The coding unit is therefore a simple multiplier; since A is a constant, its structure can be optimized to grant maximum compactness and speed. Decoding is performed, for addition and subtraction involving coded operands, by applying the linear antitransformation $N = C(N)/A$; the same decoding holds for multiplication whenever one operand only is in coded form. If both operands are coded, for a multiplication the antitransformation becomes $N = C(N)/A^2$. Whenever the antitransformation produces a non-null residue, an error is detected. Aliasing is possible whenever the error is a multiple of the code generator (A or A^2); a number of papers have been published on optimum choice of A to minimize aliasing in the various arithmetic units ([7]). In particular, it has been proved that choosing $A = 3$ gives satisfactory detection capacity for all arithmetic units in which a single fault induces an error that can be represented as the addition of $\pm 2^k$; moreover, both coder and decoders are quite simple and the bit overhead for each coded data element is just one bit. It can be noted that the above error assumption holds for all single faults if suitable design techniques are adopted for the arithmetic unit (see [8]).

A second set of codes, identified as $AN + B$, is derived from the previous one by introducing a ***code displacement*** B, that must be an odd number, prime with respect to A. The $AN + B$ code is defined as the ***associated code*** to the AN code whenever the code generator A is common to both. The $AN + B$ code is again a non-systematic code, and the arithmetic units need not be modified (excepting for the length of the operands) to perform on coded data. Coding is achieved by applying the linear transformation $C(N) = AN + B$. Decoding is obtained in acceptably straightforward manner if the operation performed is an addition, in which case, denoting by R' the result of the operation on coded data, and by R the corresponding uncoded result, it is $R = (R' - 2B)/A$.

If the operation performed is a multiplication, the antitransformation in the general case requires rather complex operations; for example, if both operands X and Y are expressed in $AN + B$ code, it is $R' = A^2XY + ABX + ABY + B^2$, so that the computation of $R = XY$ involves also the subtraction of two terms depending on the operands. Given the complexity of decoding, adoption of this new code is justified only if the performance improvement is sufficiently good. In fact, the $AN + B$ code grants that zero is not a codeword, so that no aliasing will ever occur in the presence of an error and no latency will occur in the presence of a fault. It has been proved that a code diplacement $B = 1$ is sufficient to grant the above improvements.

3. Use of $AN + B$ codes: modified neuron's architectures

We consider now the application of the coding techniques described in section 2 to the basic neuron's structure. We take into account a number of possibilities, depending on which inputs are coded, which code is adopted and what units are protected. We distinguish two alternative approaches to propagation of error information; namely, we can detect presence of an error locally on the single unit and create a separate network for propagation of error information, or propagate the error information through the nominal system structure.

3.1 Use of AN codes: local detection solutions

Let us consider first adoption of coding so as to protect "conventional" arithmetic operations (multiplications and additions); the error information generated locally is propagated by a separate network throughout the whole system. We can refer to one general modified neuron's architecture (see fig. 1) where code generators A_1 and A_2 may undergo different choices, as follows:

1. $A_1 = 1$, $A_2 = A$: weights are coded, input signals are not. We term this solution "Local Detection Coded Weight" architecture ($LDCW$). The results of the synaptic products and the sum of the weighted input signals are represented in AN code. *L*ocal *D*etection introduces a distributed checking of the computational results by verifying the correctness of the summation input to the non–linear function f. The decoder/checker generates the uncoded representation of the input summation and an error signal e : the input summation is then delivered to the non–linear function to generate the neuron's output. The result of the neuron's computation is not coded and is directly delivered to the receiving neurons.

   This architecture is capable of detecting all errors in the synaptic multipliers and in the adders performing the input summations, provided a suitable structure of the full adders has been adopted as discussed in [8]. Errors in the weight memory may be detected according to the specific value of A. If A is odd, all single errors generate non–codewords at the input of the multiplier; if the input X is divisible by the code generator A, the error remains latent until a value X which is not divisible by A is presented at the neuron input. If we assume a random distribution of the value of the neuron's input, all memory errors may be detected, possibly after a delay.

2. $A_1 = A$, $A_2 = 1$; here, we make use of Local Detection with Coded Inputs ($LDCI$), while weights are the nominal ones. The synaptic products and the sum of weighted inputs are coded in the same AN code as the neuron's inputs. Decoding, checking and error signal generation are identical to the $LDCW$ case. The neuron's output must then be encoded to propagate the coded data stream to the receiving neurons; computation of the non linear function and result encoding can be compacted in a single operation, by properly modifying the non–linear function evaluator.

   As the previous one, this architecture also is capable of detecting all errors in the synaptic multipliers and in the adders. Moreover, single stuck-at faults on transmission lines of interconnection paths between pairs of neurons can be detected, provided $| w_{ij} |_A \neq 0$; in this case, the fault generates a non–codeword. The main (and certainly not minor) drawback of this solution is the total lack of protection for weight memories.

3. $A_1 = A_2 = A$: Local Detection with Coded Weights and Inputs ($LDCWI$). Both synaptic weights and inputs are coded in the same AN code. The results of the synaptic products and the sum of the weighted input signals are thus represented in A^2N code. Local detection provides distributed checking of the computational results, as in the previous architecture. The neuron's output must be encoded to propagate the coded data stream to the subsequent neurons, as in $LDCI$ architecture.

   This architecture combines the detection capacities of both previous architectures. Error latency may occur for faults affecting the interconnection path; moreover, such faults can be detected only if $| w_{ij} |_A \neq 0$. The evaluation function is the only totally un–protected unit.

3.2 Use of AN codes: protection of the non–linear function

In all previous solutions, the result produced by the summation component is propagated (whether correct or error–affected) to the input of the evaluation function in uncoded form. We must then take into account two different issues: propagation of erroneous information through a fault-free evaluation unit (an important issue for those instances in which error masking may have been present) and insurgence of an error due to a fault in the evaluation function.

Obviously, such an analysis must take into account the form of the non-linear function and the implementation techniques adopted for it. We consider here the simplest (and, in practice, widely adopted solutions) by which the function is approximated either by a single step function or by a composition of steps. A basic assumption common to all cases is that signals be represented as fixed–point numbers, expressed with a finite and fixed number of bits.

Assume first a single step; the implementation involves a trivial comparator. Whatever the input to a fault–free comparator, by definition its value falls within the acceptable bounds for the comparator's inputs; therefore, the output will always be an *acceptable* value (i.e., one of the two alternative signal values), although it might be the *wrong* one as a consequence of an error in the input value. Even by providing coding on the output of the comparator (as in solutions $LDCI$ and $LDCWI$) no protection is given against propagating error–affected information. Errors cannot be *confined*, meaning that subsequent neurons will operate on formally acceptable input signals. It is quite obvious that local detection and local error–confinement policies are the only possible solution to this drawback.

If, on the other hand, σ is correct but the comparator is faulty, solutions of types $LDCI$ and $LDCWI$ must of necessity be considered to detect a fault in the comparator's output. The comparator must be designed so as to grant that any single fault affecting it will produce as output a non–codeword; the error may then be detected by neurons receiving such signal, following the conditions stated in section 3.1.

Consider now a multiple–step approximation. If the evaluator is correct, error propagation on error-stricken inputs happens just as in the previous case.

For faults affecting the evaluator circuit, we might envision different implementation approaches. A cascade of comparators could be considered – in which case, design requirements introduced above would still hold – or otherwise a PLA could be chosen as a compact and fast implementation. In this last case, any single fault (stuck–at on an output line or crosspoint) in the OR plane will either be masked (in the case of stuck–ats) or result in a non–codeword.

3.3 Local detection solutions: fault location network

In all solutions described in 3.1, an error signal is generated locally by the decoder detecting the error. Two approaches may be undertaken, namely, system–level error *detection* or neuron–level error *location*.

In the first case, error signals generated by all neurons in the network are OR–ed together and the result simply provides a go–no go information for the complete system. Wiring is actually complex – involving propagation of error signals from layer to layer and through all neurons of each layer. A modification, still leading to error detection only, distributes the error signal OR–ing through the whole array: this leads to lower fanin of the OR gates and to a simplified, regular wiring scheme.

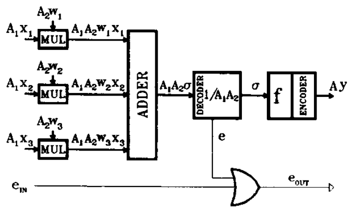

Fig. 1 - Local detection architectures for AN codes

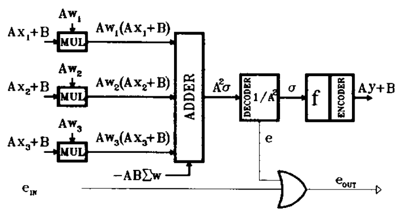

Fig. 4 - LDCWAI architecture

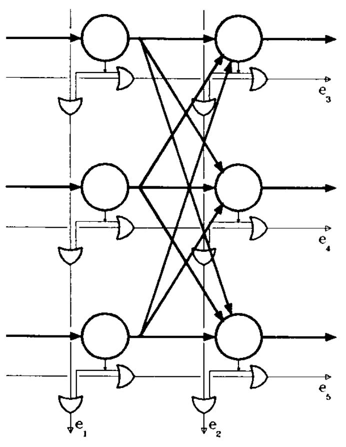

Fig. 2 - Fault location network (uncoded inputs)

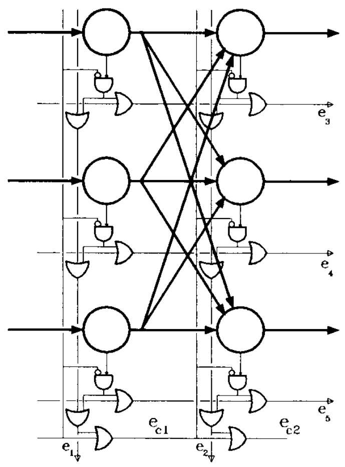

Fig. 3 - Fault location network (coded inputs)

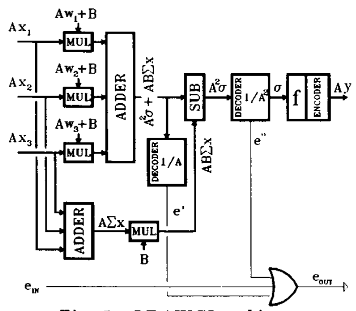

Fig. 5 - LDAWCI architecture

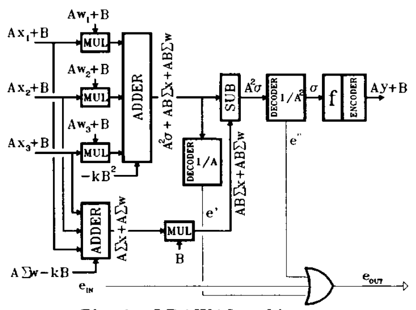

Fig. 6 - LDAWAI architecture

Error location can be provided by adopting a triangularization technique, here described in the assumption that the multi–layer network is mapped onto a rectangular array of neurons (see fig. 2). The error signal related to each neuron is now propagated along two directions, namely both along the horizontal axis (inter–layer) and along the vertical one (intra–layer). In the single–error assumption, such a technique allows to locate the faulty neuron at the crossing between the leftmost vertical error line and the uppermost horizontal error line on which an error information is present. The architecture presented in fig. 2 can be adopted only for networks in which the inputs are not coded. Otherwise, an error in the output of one neuron leads all neurons receiving such output to activate their own error signals: the layer containing the faulty neuron corresponds to the leftmost activated error line, while the neuron inside such layer cannot be identified since all horizontal error lines are active.

The architecture for fault location when inputs are coded is presented in fig. 3. Whenever the intra–layer error signal of one layer is active (i.e., an error occurred in that layer), it hinibites propagation of the error signals generated locally in neurons belonging to the subsequent layers and forces propagation of the error signals generated in the layer containing the faulty neuron. In large neural networks, the skew due to intra–layer error propagation may be relevant for overall system performances; it can be avoided by properly latching the intra–layer error signals, even if a limited error latency may be introduced.

3.4 Use of $AN + B$ codes: local detection solutions

As for AN codes, we examine first solutions protecting the synaptic errors and the summation errors, and then briefly extend them to include errors in the evaluation function.

We extend the solutions presented in section 3.1 so as to exploit the characteristics of $AN + B$ codes. Generation and propagation of the global error signal is performed as in the case of AN codes.

4. $A_1 = A_2 = A$, $B_1 = B$ and $B_2 = 0$: Local Detection with Coded Weights and Associated-Coded Input ($LDCWAI$). Synaptic weights are coded in AN code (see figure 4), while inputs are represented in the associated $AN + B$ code. The results of the synaptic products have the following composite expression:
   $(Aw_{ij})(Ax_j + B) = A^2 w_{ij} x_j + ABw_{ij}$. The coded input summation S_i to neuron i is thus: $S_i = \sum_j (Aw_{ij}) \cdot (Ax_j + B) = A^2 \cdot \sum_j w_{ij} x_j + AB \cdot \sum_j w_{ij}$.
   To verify the correctness of multiplications and additions, the input summation must be transformed into a traditional codeword. To such purpose, the term $AB \cdot \sum_j w_{ij}$ must be subtracted from S_i. Since during the recall phase the synaptic weights are fixed, such correcting term has a fixed value for each neuron. In the absence of errors, the corrected summation is $s_i = A^2 \cdot \sum_j w_{ij} x_j$, i.e., it belongs to the $A^2 N$ code. Error checking can be performed on the corrected summation as in the $LDCWI$ architecture by dividing by A^2. If an error occurs in the multiplications, in the addition or in the correcting subtraction, a non–codeword is presented at the decoder/checker whenever the same conditions as in 3.1 are assumed for the arithmetic units and the code generator A. The neuron's output must then be encoded to propagate the coded data stream to the receiving neurons.
   This architecture detects errors as the $LDCWI$ one, but no error latency may occur for errors in the weight memory since no input belongs to the AN code. Denote by S_i^* the coded input summation in the presence of an error in the weight memory. In our assumptions, the error e is an additive power of two; therefore, if $e = 2^k$, it is:

$S_i^* = \sum_j (Aw_{ij} + 2^k) \cdot (Ax_j + B) = A^2 \cdot \sum_j w_{ij}x_j + AB \cdot \sum_j w_{ij} + 2^k \cdot (A\bar{x}) + 2^k B$ where $\bar{x}$ is the input corresponding to the synapsis affected by the error. The corrected summation s_i^* in presence of such error is: $s_i^* = A^2 \cdot \sum_j w_{ij}x_j + 2^k \cdot (A\bar{x}) + 2^k B$. Even if $\bar{x}$ is divisible by A, the corrected summation does not belong to the code A^2N since $2^k B$ is not divisible by A^2.

This architecture detects also errors due to a fault in the correcting subtractor.

5. $A_1 = A_2 = A$, $B_1 = 0$ and $B_2 = B$: Local Detection with Associated-Coded Weights and Coded Input (*LDAWCI*), (see figure 5). The results of the synaptic products and the input summation have composite representations similar to those obtained for the *LDCWAI* architecture. The correcting term which transforms the coded input summation S_i into the corrected summation s_i is $AB \cdot \sum_j x_j$. Unfortunately, this is not a constant related to the values of the synaptic weights as in the *LDCWAI* solution, but is depends on all input signals of the considered layer. Since the expression is identical for all neurons in the layer, it can be computed once for each layer and, then, distributed to all neurons in the layer. Error checking can be performed on the corrected summation as previously. If an error occurs in the multiplications, in the addition, in the correction generator or in the correcting subtractor, a non-codeword is presented at the decoder/checker. To guarantee detection of errors in the interconnection paths, an additional data check must be performed at the output of the adder generating the input summation; this check consists in verifying that s_i belongs to the AN code.

   This architecture detects errors as the *LDCWI* solution as well as errors due to faults in the interconnection paths; the circuits for the correcting terms are also protected. Error latency may occur for weight memories.

6. $A_1 = A_2 = A$ and $B_1 = B_2 = B$: Local Detection with Associated-Coded Weights and Associated-Coded Inputs (*LDAWAI*), (see figure 6). The correcting term that must be subtracted from the coded input summation S_i is $AB \cdot \sum_j x_j + AB \cdot \sum_j w_{ij}$. To perform such correction, first we subtract kB^2 from the input summation S_i (where k is the number of input synapses incoming into neuron i), then we add $AB \cdot \sum_j x_j + AB \cdot \sum_j w_{ij}$. The second correcting term is obtained by adding the constant $A \sum_j w_{ij} - kB$ to the sum of the coded inputs.

   This architecture detects all single errors without any latency.

3.5 Global detection solutions

Apart from the detection performances and the cost of coding/decoding circuits, wiring has been seen to be a major cost factor in the local detection solutions previously described. Considering ***detection*** – not location – to be the only aim of the concurrent detection technique, we may envision other architectures, still based on the same coding approaches, in which the error information is propagated through the nominal architecture itself, rather than through a separate error propagation network.

We will introduce the guidelines of such modifications with reference to one architecture only, namely the *LDCI* architecture. Similar modifications can be derived in a straightforward manner from the other architectures based on coded inputs. Two solutions may be envisioned. In the first case, the components appearing in fig. 1 are left unmodified, and suitable ones are added; in the second case, the function evaluator is radically modified.

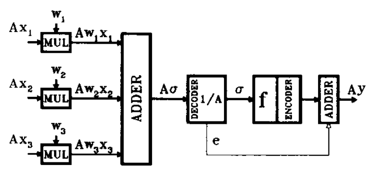

Fig. 7 - Local propagation

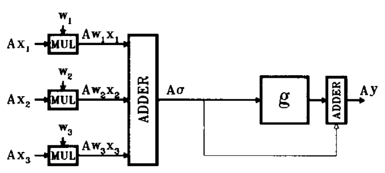

Fig. 8 - Global propagation

7. *LPCI* architecture: Local Propagation with Coded Inputs (fig. 7). The only difference with respect to *LDCI* is that the error generated locally is added to the encoded neuron's output. If no error occurs in the computation before the decoder, the output generated by the non-linear function is the codeword representing the nominal neuron's output; otherwise, the output propagated to the receiving neurons is a non-codeword.
   As for the encoder, the final adder could also be compacted into the non-linear function.
   From the point of view of error detection, *LPCI* architecture has basically the same characteristics as *LDCI*. Still, it should be noted that if the neuron's output is affected by an error but all the synaptic weights associated with its connections to receiving neurons are divisible by A, no error will be propagated and contaminated information will spread through network.
8. *GPCI* architecture: Global Propagation with Coded Inputs (fig 8). No decoding/checking is performed locally; the error is globally propagated toward the final outputs by using the nominal operations on coded data. At the final outputs of the neural network, the results are decoded and checked as in local detection. To guarantee a complete propagation of errors due both to faults in the non-linear function and to faults in the other arithmetic units, the coded representation $A\sigma$ of the weighted summation is delivered to a modified version g of the non-linear function f; we adopt $g(A\sigma) = Af(\sigma) - A\sigma$. Then, the output of g is added to the coded input summation $A\sigma$. Use of the modified function g and of the output adder can be avoided only for such implementations of the function f which guarantee that the output is a codeword if and only if the input of f is a codeword (i.e., no error occurred in circuits above f) and no error occurred in the computation of the function f itself.
   Detection performances and problems are the same as for the *LPCI* solution.

4. Comparative evaluation of costs and performances

In section 3, we evaluated the error detection performances of the different solutions; it is necessary to consider also the area overhead involved and the increase in computation time.

Consider first the relative area increase of the various components in each neuron for the different coding schemes:

- *LDCW*: Weight memories, multipliers and adder all require an increase proportional to $\lceil log_2 A \rceil$; the cost of the decoder must be added.
- *LDCI*: Multipliers, adder and wiring all require an increase proportional to $\lceil log_2 A \rceil$; the cost of the decoder and of the encoder must be added.
- *LDCWI*: Multipliers and adder require an increase proportional to $\lceil log_2 A \rceil^2$; weight

memories and wiring require an increase proportional to $\lceil log_2 A \rceil$. The cost of the decoder and of the encoder must be added.

- *LDCWAI, LDAWCI, LDAWAI*: Due to the fact that B is most often equal to 1, the cost of memories, wiring, multipliers and adder is not increased in a very relevant way with respect to the *AN* code solutions. A very relevant cost increase, on the contrary, will derive from wiring and circuits related to the various corrections involved in the decoding operations.

Last, let us refer to time overhead. From this point of view, the fastest architecture is *LDCW*; the only added operation is decoding on each neuron's adder output. Actually, a time increase in multiplication and addition should also be taken into account; again, for this solution, this increase is no higher than that for any other solutions.

LDCI adds, to the time overhead required by *LDCW*, the delay due to encoding of the neuron's output; this might be minimized or even nullified by a proper design of the function evaluator. *LDCWI* will require a higher multiplication and addition time than *LDCI*; other overheads are identical.

As for the area evaluation, also the time overhead introduced by the $AN + B$ coding solutions is quite more relevant than for the *AN* ones due to the decoding complexity.

The application's requirements and the particular structure of the network's implementation will guide in choosing the "best" solution on the basis of performances and costs as evaluated above.

5. References

[1] D.B.I. Feltham, W. Maly: "Behavioral modeling of physical defects in VLSI neural networks", in *Proc. 1990 Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Grenoble, France, Nov. 1990

[2] D.B.I. Feltham, W. Maly: "Limitation to the size of single-chip electronic neural networks" in *Proc. IEEE International Conference on WSI* , San Francisco, USA, Jan. 1991

[3] V. Piuri, M. Sami, R. Stefanelli, "Fault tolerance in neural networks: theoretical analisys and simulation results", *Proc. Compeuro 1991*, Bologna, Italy, May 1991

[4] V. Piuri, M. Sami, R. Stefanelli: "Neural networks on silicon: the mapping of hardware faults onto behavioral errors", *Proc. Int'l Workshop on Defect and Fault Tolerance 1991*, Hidden Valley, USA, Nov. 1991

[5] C. Neti, M.H. Schneider, E.D. Young: "Maximally fault tolerant neural networks", *IEEE Trans. on neural Networks*, Jan. 1992

[6] V.Piuri, M.G.Sami, D.Sciuto, R.Stefanelli: "A behavioral approach to testability of neural networks", *Proc. IJCNN92*, Baltimore, USA, June 1992

[7] T.R.N. Rao: *Error coding for arithmetic processors*, Academic Press, NY, 1974

[8] R. Stefanelli, M. Annaratone: "A multiplier with multiple error correction capability", *Proc. ARITH-6*, 1983