

On-Line Testing In Digital Neural Networks

Serge Demidenko

Singapore Polytechnic, 500 Dover Road, Singapore 139651, SINGAPORE

Vincenzo Piuri

Politecnico di Milano, Piazza Leonardo Da Vinci 32, Milano 20133, ITALY

Abstract

On-line testing is a basic issue of any concurrent fault-tolerance policy. Error localisation within the neural network is necessary to provide information for hardware reconfiguration in order to achieve the system survival. In this paper, a concurrent approach for error localisation in digital neural networks is discussed and evaluated. Two techniques are applied: concurrent diagnosis with the use of data coding for error detection at neuron level and on-line localisation of the faulty neuron within the network.

1. Introduction

The use of *Artificial Neural Networks* (ANNs) in mission-critical applications implies adoption of ad-hoc techniques to provide fault tolerance. Concurrent error detection allows to guarantee continuous checking of results; concurrent diagnosis is required to localise the faulty components as fast as possible in order to minimise the time for system repairing. System survival can then be achieved by replacing the faulty components in the active computation (e.g., by means of reconfiguration), if enough redundancy has been provided.

In this paper an approach to concurrent testing in ANNs at functional level is discussed, by improving some preliminary results reported in [1]. Attention is focused on digital implementations of multi-layer feed-forward networks [2], since they have wide application areas (in particular, in signal and image processing). The goals are to achieve concurrent error detection in the neural computation, and to localise the faulty neuron within the network. The error detection is performed by means of traditional data coding techniques suited to the specific neuron implementation. Localisation of faulty neurons is performed by using on-line testing techniques by identifying the layer containing the faulty neuron, and by localising of the faulty neuron within such a layer.

2. Neural And Error Models

In this paper, we consider a widely adopted neuron's model [2] defined by
$$Y_j = f_j \left(\sum_{i=1}^m w_{ij} x_i - \Theta_j \right), \quad (1)$$

where m is the number of inputs $\{x_i\}$ of the j -th neuron; Y_j is its output; f_j is its activation function; w_{ij} is the synaptic weight between the input of the neuron j and the output of the neurons i ; and Θ_j is the threshold.

Our analysis is performed on multi-layered feed-forward ANNs composed by such neurons (fig. 1). However the basic results may be extended to other classes of neural paradigms that can be mapped onto similar hardware architectures.

Direct digital implementation of equation (1) involves one-to-one mapping onto digital components: weights are stored in memory elements, synaptic products are evaluated by digital multipliers, summation is performed by an adder, and a suitable circuit computes the non-linear function. For simplicity's sake, we restrict our analysis to bit-parallel arithmetic units.

The fault model we adopt is based on the general structure introduced above: faults may occur within the components of the neural architecture (weight storage, arithmetic devices, non-linear function evaluator). Faults in the interconnections may also be taken into account if they induce errors identical to faults in the components. A functional error model is defined to describe the erroneous computation induced by occurrence of faults in ANN [3]. Errors taken into account can belong to any of the subsequent classes:

- *synaptic errors*: defined as producing an unexpected value of the weight-signal product. These errors may be due to an error affecting either the synaptic weight or the associated multiplication; in the absence of any implementation detail, these two instances must be considered as indistinguishable;
- *summation errors*: they affect the summation of the weighted inputs within the individual neuron;
- *errors affecting the non-linear evaluation function* within the individual neuron; no particular assumption is introduced concerning the form of the erroneous function. This implies an erroneous value is forwarded to all neurons connected to such neuron's output.

In this functional error model, the *single error* is any erroneous neural result (induced by any hardware faults considered here) which affects only the output of *one* neuron. For simplicity's sake, we consider here only the single error case.

3. On-Line Error Detection And Localisation

On-line testing in ANN can be achieved by:

1. Detection of the error presence within a neuron
2. Identification of the layer containing the faulty neuron
3. Localisation of the faulty neuron within such a layer.

The general scheme implementing this approach is shown in fig. 2. Each neuron includes an architectural support for concurrent error detection. Whenever a fault induces an error in the computation, the error signal generated in the faulty neuron becomes active.

For the neurons belonging to a layer, a diagnostic word (*signature*) is generated by compressing the information associated to the neurons' error signals. When a fault occurs an erroneous signature (i.e., a signature different from the reference value) is generated for the layer containing the faulty neuron. Since we have not adopted any error confinement policy to limit the circuit complexity, and since the erroneous data may be distributed and used several times in subsequent layers, also the signatures computed for such layers may be erroneous. The neural computation proceeds from the input layer towards to output one. Therefore the faulty neuron belongs to the layer which has an erroneous signature and which is the nearest to the input layer.

Propagation of the erroneous signature is realised by means of the propagation block towards the external environment; it delivers the signature and the identifier of the layer containing the faulty neuron.

3.1. Error Detection

Neuron's architecture can be enhanced to support concurrent error detection. Modular redundancy with outputs' comparison is conceptually very simple but has an impressive cost in terms of silicon area. Typically, the circuit complexity increase is about 110%, while an additional silicon area for routing could be up to 20% of the device area. At the same time the latency is quite low (less than 2%). Some algorithmic approaches to achieve modular redundancy can be considered [4]. However they are often too application specific, or require restructuring and redefinition of the neural operations.

Data coding techniques give in general attractive results without affecting the computational performance [3, 5]. Besides, they do not require any modification of the neural operation (learning, recall, generalisation). Focusing on *AN codes* and on *residue codes* is justified since they are cost-effective in terms of structure redundancy and detection capability.

AN codes are *non-systematic codes* where each nominal datum N is replaced by its coded representation $C(N)$ via the linear transformation $C(N) = A \times N$, where the integer constant A is the *code generator*. The overall structure of the arithmetic units (adders, subtractors, multipliers) is not modified by the adoption of the coded data; the only difference is the number of bits required, that obviously increases with A . Since A is a constant, the coding unit can be optimised to grant maximum compactness and speed. For addition and subtraction

involving coded operands, decoding is performed by applying $N = C(N)/A$; the same decoding holds for multiplication whenever one operand only is coded. If both operands are coded, it becomes $N = C(N)/A^2$.

The structure of the neuron with concurrent detection capabilities by using *AN codes* is shown in fig. 3. Whenever the decoding produces a non-null residue, an error is detected.

AN codes are capable of detecting errors in synaptic multipliers and in input adders, if a proper design of these components is adopted. Single errors in the weight memory can be detected if the weights are coded by using any odd A . Error aliasing may occur if the corresponding neuron's input is divisible by A . If neuron inputs are coded, single stack-at faults on transmission lines of interconnection paths between pairs of neurons can be detected. Aliasing occurs if weights associated with the faulty interconnections are divisible by A . As the evaluator of the non-linear function is concerned, error detection is strictly related to the adopted implementation: due to non-linearity, *AN codes* cannot in fact usually be exploited [3].

It has been proved that $A=3$ provides complete detection for arithmetic units in which faults induce any single binary error. Both the coder and decoder are very simple. Since the checking is performed on the weighted summation, the $3N$ coding guarantees detection of a larger class of faults: any multiple fault appearing as a single bit error in the weighted summation can in fact be detected since it is not a multiple of A . The circuit complexity and latency increases are varying with the length of the operands. Typically, in the case of multipliers having a single coded input, both these parameters are around 27% for 8-bit parallel operands, and are decreased to about 15% for 16 bits. For multipliers with both coded inputs, the above two figures become 50% and 30%, respectively.

AN + B codes (that are an extension of *AN codes*) provide improved detection capabilities. In fact, these codes remove the null codeword. Arithmetic operations and architectures are similar to the previous ones, even if additional circuits must be considered to deal with the *code displacement B* [2]. However the use of these codes can be discarded due to the increase both in system complexity and latency.

In residue codes [5], each nominal datum N constitutes the information bits, and the redundancy bits that are given by the residue $R_b(N)$ of the datum N modulo b , where b is the *base* of the code. The coded representation $C(N)$ is obtained by using the bit concatenation $C(N) = [NR_b(N)]$. An accurate choice of the base must be considered to achieve the highest detection capability at the minimum cost. It has been shown that $b=3$ gives complete detection for all arithmetic units where a single fault induces an additive error [6]. As in the case of *AN codes*, these codes actually are able to detect the presence of any multiple fault occurring in the circuit related to the computation of the weighted summation within each neuron, whenever it appears as a single-bit error in the weighted summation itself.

Residue codes are *separate*: the computation on the nominal datum can be performed in parallel to the computation on the redundancy bits. The arithmetic units (adders, subtractors, multipliers) operating on the information bits are not modified by the adoption of the coding scheme. The units processing the redundancy bits must be capable of operating in the modulo arithmetic. The coding unit computes the residue modulo b of the uncoded data, while decoding is not required since the nominal output can be directly extracted from the output codeword by dropping off the redundancy bits.

Result checking consists of comparing the residue of the nominal output to the bits generated from the redundancy bits of the input operands. Aliasing occurs if the residue modulo b of the error is null. Residue codes have the same detection capabilities as the AN codes with respect to the error model introduced above in section 2.

The circuit complexity increase for this class of codes is slightly higher in comparison with the case of AN codes. It is about 45% and 20% for correspondingly 8- and 16-bit multipliers with a single coded operand, while it is about 5% higher for multipliers with two coded inputs. Conversely, the latency increase (due to encoding and checking) is not substantial - about 10%.

The choice of the solution which is the best suited to a given application must be performed by considering the actual architectural and implementation technologies, i.e., the actual evaluation of the design parameters.

3.2. On-Line Localisation

Localisation of the faulty component in the ANN may be performed concurrently with the nominal operations whenever the application is so critical as to require immediate reconfiguration of the system to guarantee the maximum availability. Different techniques can be adopted to achieve the goal. In [2] a triangularization approach is proposed for highly-regular feed-forward ANNs implemented as a modular array structure. However, the solution may present some drawbacks concerning the regularity and modularity of the structure for networks composed by different-sized layers.

The use of on-line compact testing (testing with compressed data) is more general. It provides a structured approach to error localisation and, thus, to identification of the faulty components. The solution is based on the adopted error model, and the single error assumption at the neuron level. The error bits generated by the layer neurons constitute the input stream being processed (compact). The neuron error signal is 0 if no error is detected, the signal is 1 otherwise. The input sequence for the layer data compressing block is composed of 0's only if no error has been detected in the layer, and the layer error signature is null. If a fault induces an error, at least one member of the sequence is 1, and the actual signature is not null (if no aliasing masks the error). Under the assumption of single errors, not more than one member of the sequence is equal to 1 at time.

There are specific and partially conflicting requirements in choosing the compression technique

suited for a given application. First of all, the degree of the compression must be high while the complexity of the algorithm must be reasonably modest in order to minimise the hardware overhead/latency. On the other hand, the compression should not lead to error masking with respect to the adopted error model. Moreover any single erroneous neuron should be uniquely identified by the signature computed on the actual data.

Application of the on-line signature compression based on a distributed architecture of a linear feedback shift register is presented in [1]. The solution is rather effective in terms of the hardware overhead, and the space on the silicon: a limited number of the interconnection lines and logic gates are in fact used to implement the error data compressor. However the localisation of the erroneous datum (and thus the faulty neuron) is associated with either substantial external diagnostic overhead or significant computational delay.

In this paper we adopt another approach that still satisfies the above requirements. It is based on the use of an encoding circuit in each ANN layer in order to generate the address of the faulty neuron. The error bit of each neuron of a layer is connected to the corresponding input of the encoder. If the bit is equal to 1, the encoder output gives the binary number (address) of the neuron where the error has been detected. If no error occurs in the layer, the encoder output code is null. The solution provides effective on-line localisation of the erroneous neurons. However it leads to the substantial number of the interconnection lines and thus to the excessive space occupied on the silicon.

To preserve the architecture regularity, and to minimise the routing area it is necessary to avoid the adoption of complex schemes with numerous interconnection lines. The error data compression block can therefore be realised by means of a distributed architecture. An example of such a realisation for the case of 10 neuron layer is shown in fig. 4. It is obtained by unrolling the parallel encoder structure into the serial one. The encoder is composed of a cascade of OR gates that are interconnected in a linear chain according to the encoder logic scheme. It is easy to see that the number of the interconnection lines along the layer is a constant not exceeding the smallest integer $\geq \log_2(M+1)$, where M is a number of the neurons in the layer.

To reduce the time for the generation of the signature (the address of the neuron in the layer where an error has been detected) a tree-like organisation of the encoder can be adopted. This allows to exploit the intrinsic computational parallelism of the tree-like scheme: the linear chain is partitioned into sub-chains while the signals in the sub-chains are processed in parallel.

Identification of the layer containing the faulty neuron can be performed by analysing the set of signatures generated by all layers. The signatures are tested starting from the one produced by the input layer, and proceeding towards the output layer: the faulty layer corresponds to the first one having a non-null signature. This approach is practically realisable only if the total number of bits for all signatures is reasonably low.

When the number of signature bits is high but the number of layers is low, we compress the identification information into the layer error signals: each of them is obtained by OR-ing all signature bits at the output of the corresponding layer compression block. These signals are extracted from the architecture through separate lines. They are tested starting from the one produced by the input layer and proceeding towards the output layer. The faulty layer is identified by the first signal which is active. The signatures are multiplexed on the same lines in order to deliver the data localising the faulty neuron to ANN's diagnostic output.

If the number of layers is high, the following approach is employed. Each layer is identified by its *layer address*. The address of the faulty layer is propagated in parallel with the fault signature. The basic block of the signature propagator is shown in fig. 5. If the control signal is 1, the signature generated in the layer and the local address are propagated to the previous layer; otherwise, the signature and the address received from the subsequent layer are forwarded to the previous one. The multiplexer control signal is the layer error signal (it is 0 if no signature bit of such a layer is 1). The structure guarantees correct and unique identification of the faulty layer and the faulty neuron within such a layer.

To reduce the computational delay, a tree-like organisation for the propagator can also be adopted.

4. Evaluation of Hardware & Time Overhead

In this paper, we analyse only the overhead due to diagnosis and fault localisation (i.e., due to the circuits for the signature generation and propagation).

First of all, we evaluate the silicon overhead due to the compression blocks. To simplify the analysis, we focus our attention on a single layer: results are valid for each layer of the network. In the case of the distributed linear encoder the number of OR-gates is at most 2^{n-1} , where n is the smallest integer number $\geq \log_2(M+1)$, and M is the number of the neurons in the layer. The number of interconnection lines between the adjacent neurons within a layer is constant, and is equal to n .

The hardware redundancy due to the propagation block is related to the architectural solution adopted. In the case of a linear structure, it requires $L-1$ 2-input multiplexers, and $L-1$ OR-gates to generate the multiplexer control signals (L is the number of the layers). In the case of a tree-like organisation, the hardware overhead for the active devices is identical to the linear case, while interconnections may require a larger silicon area.

The second figure of merit is the computational delay τ_d introduced by concurrent diagnosis with respect to the nominal neural computation. It is:

$$\begin{aligned}\tau_d &= \max_{i=1}^L (\tau_{S_i}) + \tau_p - \tau_N \\ &= \max_{i=1}^L \left(\sum_{h=1}^i \tau_{N_h} + \tau_{G_i} \right) + \tau_p - \sum_{h=1}^L \tau_{N_h},\end{aligned}$$

where τ_{S_i} is the time required to achieve steady values at the output of the signature generator for the i -th layer after presentation of ANN's inputs, τ_p is the

computational time of the signature propagator, τ_N is the computational time of the nominal network, τ_{N_h} is the computational time of layer h , and τ_{G_h} is the computational time of the signature generator.

The computational time τ_{G_i} due to the linear signature generator (distributed linear encoder) within i -th layer is proportional to the number of cascaded OR-ing, and is equal to $M/2$ times of the OR-gate propagation delay.

The computational time for the tree-like generator is due to OR-stages of a sub-chain, and due to the gates for sub-signature composition. The contribution of the sub-chains is proportional to $M/2k$, while the contribution of the composition gates is $\log_2 k$ times the propagation delay of OR-gate. Therefore, the total delay is $M/2k + \log_2 k$ times the propagation delay of OR-gate.

The computational delay introduced by the propagation block is related to its structure. In the case of the linear propagator, the delay is due to the multiplexing stages of the propagation chain. In the worst case, the actual delay is equal to the delay time of the whole propagation chain, i.e., it is $L-1$ times the delay introduced by a multiplexer.

In the case of the tree-like structure, by assuming a complete tree organisation having $k = L/2$ blocks at the first level, the computational time is $\log_2 L$.

5. Dynamic Reconfiguration

Information generated by on-line diagnostics identifies uniquely the faulty neuron. This data is used to provide the system survival by excluding the faulty component from the computation. Dynamic redundancy may be implemented whenever survival capability of the system is required, in particular for critical applications or when maintenance is difficult or impossible.

To achieve the goal, additional features must be introduced in the architecture on which the neural network is mapped. First of all, a regular, reconfigurable architecture must be adopted: redundant neurons and interconnections must be added to the basic structure as spare components which can be used to replace the faulty ones. Whenever a neuron or an interconnection is recognised to be faulty it is excluded from the nominal computation and replaced by a spare component. For example, if a regular array structure with switched busses is adopted [7], replacement is achieved by reconfiguring the interconnection switches [8].

If a bidimensional array of processing elements is used to implement the neural network, neurons belonging to the same layer can be mapped on processing elements placed in the same array column. Spares can be added in each column so that the interconnections can be switched to exclude the faulty neuron and to include one of the spare: a column fault-stealing technique [8] is effective within each column to achieve reconfiguration.

A more general mapping and reconfiguration strategy may be envisioned to deal with the irregular implementation due to the different sizes of the layers. General bidimensional fault-stealing [8] is effective if the neuron structure is large enough to justify the use of a bidimensionally reconfigurable switch interconnection

structure. Spares can be placed on the array borders or distributed within the array itself, while the interconnection network is not differentiated along the columns or between adjacent columns in order to support replacement schemes through the whole array.

If the internal structure of neurons is different (e.g., due to different number of interconnection weights), the spares must also be different: they should be of the same type as the specific neuron they must replace, or they need to be adaptable to exclude unnecessary inputs from the weighted summation computation.

An alternative structure is based on the use of equal-size pseudo-neurons capable to perform only part of the weighted summation or even the non-linear activation function: each neuron is implemented by using as many pseudo-neurons as necessary to hold all the interconnections. This approach allows to reduce the possible oversizing of the spares to guarantee complete adaptability, and consequently to limit the increase of the network complexity.

All the above approaches suffer for the problem related to the availability of the suited interconnection weights in each neuron after reconfiguration. If the functionalities of a neuron have been moved from a processing element to another one, also the interconnection weights must be moved or reloaded into the new processing element. As a consequence, the computation of the whole network must be stopped, the weights must be reloaded, and the computation be then restarted after intermediate data clearing. This implies the availability of storage within each neuron, and the possibility of external downloading through dedicated interconnections and control signals.

In the case of hardwired weights, the network complexity is very high since each processing element must hold not only one set of interconnection weights, but all possible weight sets of the neurons that could be mapped on such an element. The reconfiguration is also difficult: the active set is selected through multiplexers.

The switch interconnections are controlled by suited control signals. Due to their high number, all of them cannot usually be directly connected to external pins. As a consequence, on-board memory elements must be introduced to hold the current configuration of each switch: the stored value is used as control signal for the switch itself. Due to the limited area occupied on silicon by the switches and the memory elements, they are usually considered fault free.

Monitoring the error information, and reconfiguration control must be accurately considered to achieve an efficient and effective system survival. Generation of the reconfiguration control signals can be performed by adopting an on-board concurrent solution whenever high system availability is mandatory for the application envisioned. In this case, the reconfiguration control subsystem uses the previous distribution of faulty neurons within the architecture and the position of the new faulty neuron to identify a new mapping of the neural paradigm onto the hardware. If such a mapping exists, the corresponding control signals are stored in the on-board

memory element controlling the switches and applied to the interconnection structure; the network is still able to continue its full operation. Otherwise, the network functions may be degraded till a completely erroneous behaviour.

In the case of complex interconnection structures, an effective reconfiguration policy cannot often be computed on-board by using a dedicated subsystem having reasonable dimensions. To deal with these cases and when a very fast reconfiguration is not mandatory, an off-board approach can be adopted to select the suited mapping for the actual fault distribution. Control signals are then down-loaded onto the neural architecture to achieve the operating reconfiguration.

6. Conclusions

In this paper an approach to on-line testing of artificial neural networks has been presented. The basic structure of the digital network has been enhanced by introducing circuits for concurrent error detection based upon arithmetic codes. Additional components for compressing the neuron error signals into a short word - signature, and for propagating the layer signatures towards external environment have been discussed. Concurrent localisation of the faulty neuron is then achieved by observing the results of the error compression and propagation. By using the produced information, it is possible to design an architecture capable of on-board diagnosis and automatic reconfiguration after fault detection and location. This could be useful to minimise the diagnosis time and the repair time, and, thus, to maximise the system availability.

References

1. Demidenko S., Piuri V., Sami M., Stefanelli R.: "Fault-Tolerant Neural Architectures: a General Approach to Concurrent Diagnosis Based on Signature Analysis", *Proc. IJCNN'93*, Nagoya, Japan, 1993
2. Hertz J., Krog A., Palmer R.G.: *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991
3. Piuri V., Sami M., Stefanelli R.: "Arithmetic Codes for Concurrent Error Detection in Artificial Neural Networks: the Case of AN+B Codes", *IEEE Proc. Workshop on Defect and Fault Tolerance in VLSI Systems*, Dallas, TX, 1992
4. Piuri V.: "An Algorithmic Approach to Concurrent Error Detection in Artificial Neural Networks", *IEEE Proc. Int'l Workshop on VLSI Signal Processing*, Napa, CA, October 1992
5. Piuri V., Villa M.: "Residue Codes for Concurrent Error Detection in Artificial Neural Networks", *Proc. IJCNN'93*, Portland, OR, July 1993
6. Rao T.R.N.: *Error Coding for Arithmetic Processors*, Academic Press, NY, 1974
7. Distanti F., Sami M., Stefanelli R., Storti-Gajani G.: "Fault Tolerance in Massively Parallel Silicon Architectures: the Case of Neural Nets", *IEEE Proceedings*, April 1991
8. Negrini R., Sami M., Stefanelli R.: *Fault tolerance through reconfiguration in VLSI and WSI arrays*, MIT Press, 1989

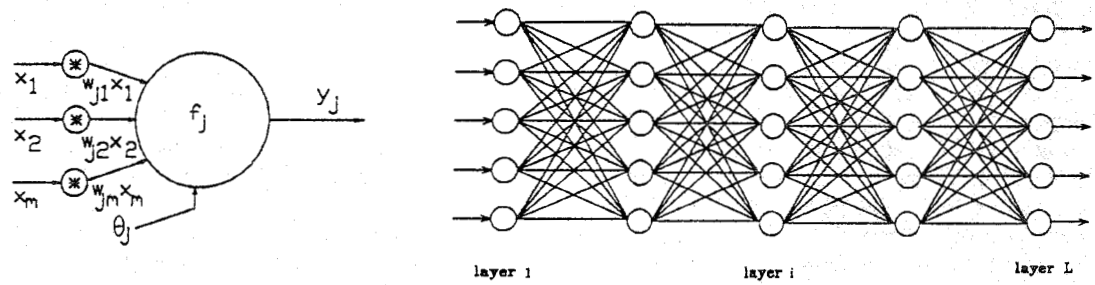


Fig. 1 - A multi-layered feed-forward neural network: the neural model and the network structure

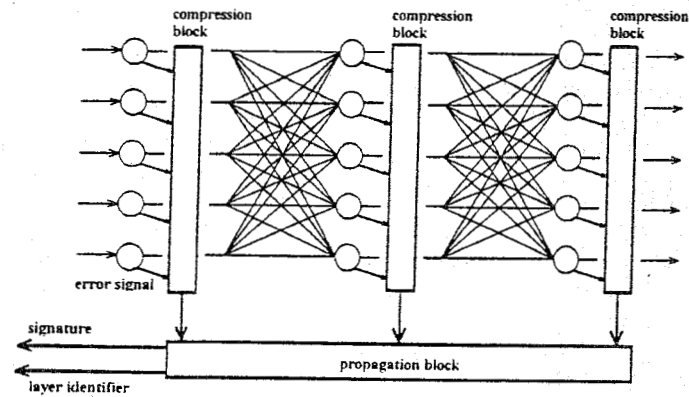


Fig. 2 - The general structure for concurrent diagnosis

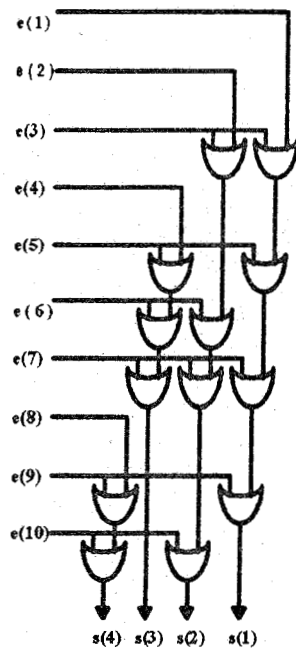


Fig. 4 - The layer signature generator

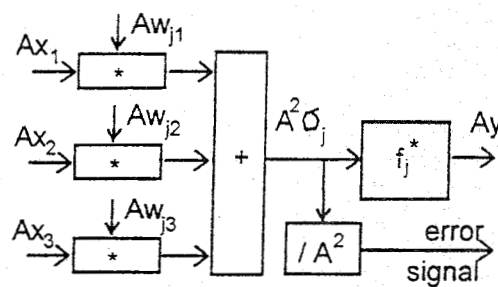


Fig. 3 - The neuron with concurrent detection capabilities

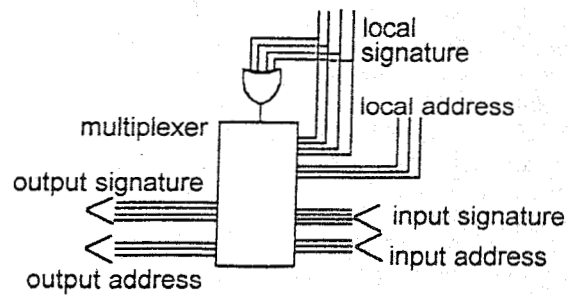


Fig. 5 - The signature propagator block