

Time-redundant Multiple Computation for Fault-tolerant Digital Neural Networks

Yuang-Ming Hsu^{*}, Vincenzo Piuri[§], and Earl E. Swartzlander, Jr.^{*}

^{*} Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712, USA
ymhsu@pine.ece.utexas.edu, e.swartzlander@compmail.com

[§] Department of Electronics and Information
Politecnico di Milano
piazza L. da Vinci 32, I-20133 Milano, Italy
piuri@elet.polimi.it

Abstract

In mission-critical applications of artificial neural networks, error correction at the architectural level is often mandatory to guarantee consistency and reliability of the network's outputs. Time redundancy allows for fault tolerance with low circuit complexity overhead. In this paper, the application of REcomputing with Triplication With Voting (RETWV) at the system level is proposed for concurrent error correction in neural networks. Feed-forward multi-layered neural networks are considered as an example, but the proposed technique can be easily extended to different neural paradigms.

1. Introduction

Neural networks are an attractive solution for massive computation in several application areas. Advances in integrated circuit technology and architectural design allow implementations at reasonable costs. The use of neural architectures in mission-critical applications (e.g., in aerospace environments and critical control systems) implies the need to guarantee the consistency and reliability of the outputs generated by the underlying architecture.

The intrinsic correction capability of some neural paradigms is often claimed as an obvious solution to the fault-tolerance issue; however, it does not cover some relevant error classes and it is not present in several neural paradigms. Besides, even in the networks trained to provide intrinsic error masking, it does not provide any information either on the actual correctness of the outputs or the location of the faulty component. Statistical analysis [1, 2] gives only the probability that the output is correct even in the presence of faults in the individual neural operations. It is not able to certify the correctness of every network result. Moreover, it is not related to the time or space distributions of faults and

errors so that the actual correctness of the network operation cannot be deduced and the exhaustion of the intrinsic error masking capability cannot be forecasted.

To guarantee error detection and correction, it is therefore necessary to adopt other techniques. In the literature, efficient algorithm-based approaches have been proposed for neural computation [3, 4], but they are either too application-specific or quite expensive in terms of circuit complexity. The use of several data coding techniques [5-7] has also been used successfully for time-efficient implementation of digital architectures, even if the circuit complexity may increase the realization difficulties.

In this paper, we consider the use of time redundancy to achieve fault tolerant digital neural networks with low circuit complexity. Feed-forward multi-layered neural networks are considered as an example, but the proposed approach can be adopted for any other kind of neural paradigms.

Section 2 describes the neural model, the related error model, and the general architectures that can be adopted to implement neural paradigms. Section 3 introduces concurrent error correction using time redundancy. Section 4 presents the neural network architectures supporting RETWV for concurrent error correction. Finally, a conclusion summarizes the discussion.

2. The neural computation and the digital architectures

Artificial neural networks are computational models capable of exploiting massive computational parallelism through holographic distribution of information and computation in the whole system [8], even in the presence of incomplete or noisy data. Neural networks contain a high number of computing elements (*neurons*), which are (often densely) connected by weighted interconnections (*synapses*) [8]. The general

model of the computation performed by neuron i is shown in Fig. 1 and can be described as:

$$y_i = f_i(\sigma_i - \theta_i) = f_i(\sum_k w_{ik} x_k + w_i i_i - \theta_i)$$

being i_i the input of the neuron from the external world, x_k the neuron's input from neuron k , w_{ik} the interconnection weight from neuron k to neuron i , w_i the interconnection weight for the possible primary input, y_i the neuron's output, θ_i the threshold, σ_i the weighted summation of neuron's inputs, and f_i a nonlinear activation function. Therefore, the neuron's computation is basically the application of a nonlinear function to the inner product of the neuron's inputs. And the generated output is delivered as input to other neurons or the network's final output.

We do not impose any restriction on the nature of the nonlinear function. We assume that operands and neuron weights are n -bit integers or fractional real numbers in two's complement representation.

Several network topologies have been proposed in the literature [8]; they can be classified as multi-layered structures, cyclic networks, and amorphous systems, according to their interconnection structure. In this paper, we focus our attention on the multi-layered feed-forward networks (Fig. 2) as an example to illustrate the proposed techniques. In multi-layered networks, neurons are partitioned into disjoint sets L_j (called *layers*): neurons in layer L_j receive inputs only from neurons in layer L_{j-1} , without any feed-back loop or intra-layer connection.

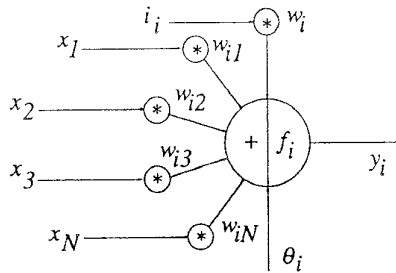


Fig. 1 - The neuron model.

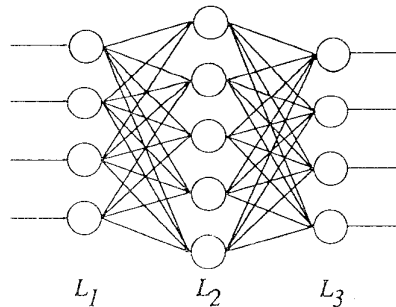


Fig. 2 - The network model.

The detail configuration of the interconnection weights for the specific application problem is determined by the *learning* procedure. Several techniques have been presented in the literature [8]. Even though the traditional approach for multi-layered feed-forward network is based on the back-propagation technique, it is not relevant to our fault-tolerance techniques since learning operates on the neural behavior while our techniques are applied at the underlying architectural level. The error correction capability of our technique can be enhanced by adopting a suitable learning procedure to achieve an intrinsic high level of error masking capability in the neural paradigm. We consider only dedicated networks which have been completely customized for the specific application, i.e., the learning has been completed off-line and weights are fixed.

A direct implementation of a neural paradigm implies a one-to-one mapping of neural operators onto the digital components (Fig. 3): weights are stored in registers, products $w_{ij}x_j$ and $w_i i_i$ are evaluated by digital multipliers, summations are performed by adders, and suited circuits generate the neurons' output from the weighted summations. The basic architecture of this kind was proposed in [9]. To increase the system throughput, pipelining can be adopted. Another implementation is to use time-multiplexing so that some components can be reused to reduce the overall circuit complexity. For the sake of simplicity, we consider in this paper only bit-parallel arithmetic units, with operands supplied at the same time; however, the techniques can also be applied to bit-serial or serial/parallel arithmetic units.

No specific implementation technology is assumed here, therefore, no fault model can be envisioned. To describe the influence of faults on the neural computation, a traditional error model at the function level is used [10]. Our goal is to correct any transient or permanent errors in the result of any single arithmetic

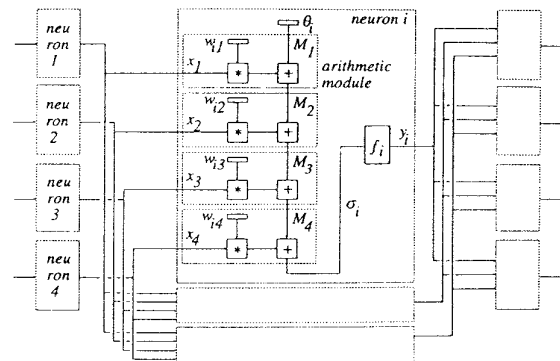


Fig. 3 - Implementation of the neural network using one-to-one mapping.

unit and in interconnection paths, whenever faults induce errors in only one component. Any error multiplicity (i.e., any number of bits affected by the error within the erroneous result) is also covered.

3. Concurrent error correction using time redundancy

Different approaches have been presented in the literature for time redundancy [11-14]. Basically, all of them repeat the same computation several times by using the system components in a different way for each repetition. Then, the results are compared to detect the presence of errors and, whenever possible, to correct the errors.

REcomputing with Duplication With Comparison (REDWC) proposed by Johnson [13] for concurrent error detection in adders has been effectively extended to concurrent error correction in simple arithmetic units (adders and multipliers) [14] using REcomputing with Triplication With Voting (RETWV). In this approach, the operands are split into three parts and the three semi-operands are mapped onto three identical sub-circuits that replicate the same computation. The least-, middle-, and the most-significant parts of the split operands are used during the first, second, and third phase, respectively, while registers hold the intermediate carries that propagate through the phases. The correct result is generated by majority voting. REDWC and RETWV have been shown very cost-effective in complex arithmetic units (e.g., inner product units and convolvers). The use of RETWV technique in digital neural architectures at the neuron level is proposed in this paper to achieve concurrent error correction.

Using REDWC for concurrent error detection can also be treated in a similar way by dividing the operands and circuits into two parts and replacing voters by comparators.

4. The RETWV neural architecture

The architecture for the one-to-one mapping of neural operators on arithmetic components with concurrent error correction using RETWV is shown in Fig. 4. The neuron inputs x_k are split into three parts: x_k^L , x_k^M , and x_k^H , each having $\lceil n/3 \rceil$ bits. The arithmetic units within M_k are divided into three parts and operate in parallel on the weights and the reduced-size neuron inputs. The selection of the corresponding parts in x_k for each step of RETWV is provided by multiplexers in the neurons generating x_k . The bit positions of threshold θ_i input is also aligned by multiplexers. Regularity of the architecture is preserved by using identical adders for partial summations with $n + \lceil n/3 \rceil + \lceil \log_2 N \rceil$ bits.

The operation of the neuron computation is performed in three steps. In the first step, x_k^L are multiplier by w_{ik} in three identical $\lceil n/3 \rceil$ -by- n array multipliers in parallel and the partial products are added by the lower-significant bits of θ_i . The partial result (σ_i^L) is stored in the registers of the merging module. In the second steps, x_k^M are multiplier by w_{ik} , and the partial products are added by the middle-significant bits of θ_i . The partial results (σ_i^M) are also stored in the registers of the merging module. The operation in the third step is the same except that the third partial result (σ_i^H) is not stored but fed directly to the final merging adder. The merging module adds the partial summations with proper shifting to generate the final weighted summation σ_i . The majority voter corrects (or masks) errors in the linear computation of steps L , M , and H . At the end of step H , the voter identifies the correct value among the three results generated by the merging module.

A single voter at the output of the whole linear part cannot locate the faulty module within the neuron. This can be improved by comparing two pairs (out of the three possible pairs) of each M_k 's three outputs at each step. If both of these comparators indicate no mismatch, no error occurred in the computation of any module M_h , where $h \leq k$. The faulty module is the one having the smallest value of k among those modules with active error signals.

The nonlinear activation function f_i can be triplicated and voted as shown in the figure to guarantee correct operation. The voters in the figure are assumed to be fault-free. However, self-checking voters and comparators can be used to indicate faults in themselves. The checking signals generated by checkers and voters in the neuron are merged to generate a single error signal for the whole neuron. An error is detected in the neuron whenever any of the checking signals has been active during the RETWV steps.

5. Concluding remarks

A new approach to design neural networks with fault-tolerance capability at architectural level has been presented. Concurrent error correction can be achieved by using time redundancy based on RETWV. This technique is effective for error masking with a low increase of circuit complexity. Latency in the error-correcting architecture without pipelining is about three times the original; throughput is one third of the original. However, the throughput can be increased to about 115% with respect to the nominal one when pipelining is adopted. This is because the fault-tolerant structures benefit from the reduced operand size and thus reduced computation time.

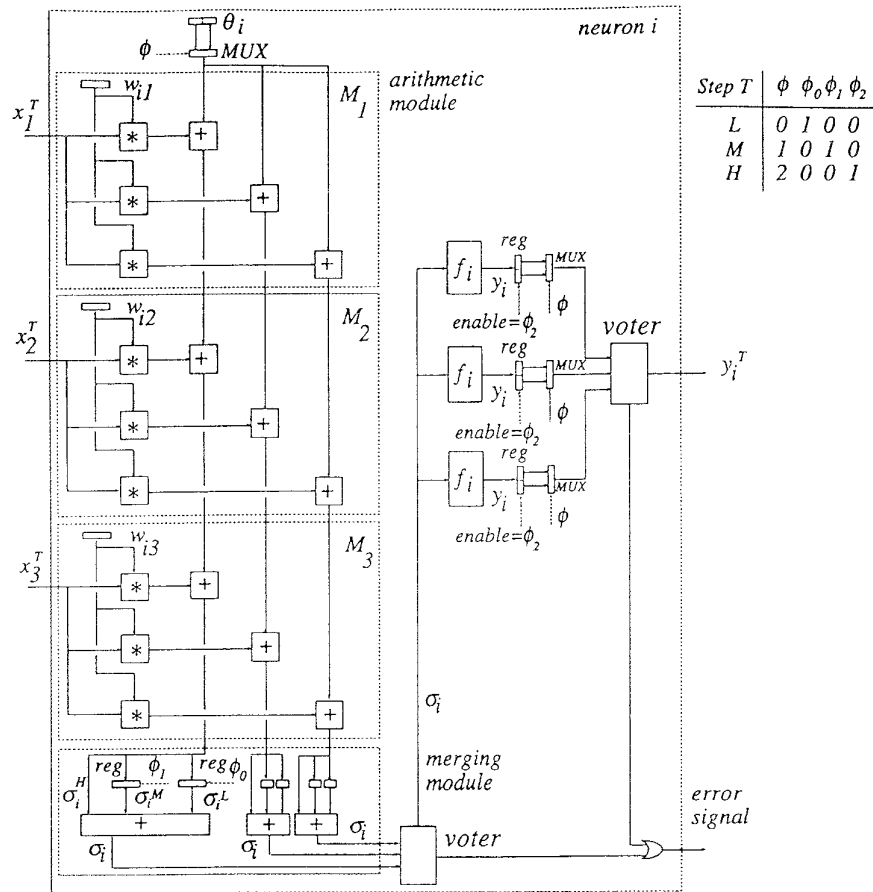


Fig. 4 - Implementation of the neuron using RETWV technique.

References

- [1] M. Stevenson, R. Winter, and B. Widrow, "Sensitivity of feedforward neural networks to weight errors," *IEEE Trans. on Neural Networks*, Vol. 1, N. 1, March 1990.
- [2] C. Alippi, V. Piuri, and M. Sami, "Sensitivity to errors in artificial neural networks: a behavioral approach," *IEEE Proc. ISCAS'94*, London, UK, June 1994.
- [3] V. Piuri, "An algorithmic approach to concurrent error detection in artificial neural networks," *IEEE Proc. Int'l Workshop on VLSI Signal Processing*, Napa, CA, Oct. 1992.
- [4] L. Breveglieri and V. Piuri, "Error detection in digital neural networks: an algorithm-based approach for inner product protection," *Proc. 1994 SPIE Conference - Advance Signal Processing*, S. Diego, CA, July 1994.
- [5] V. Piuri, M. Sami, and R. Stefanelli, "Arithmetic codes for concurrent error detection in artificial neural networks: the case of AN+B codes," *Proc. 1992 IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Dallas, TX, pp. 127-136.
- [6] V. Piuri and M. Villa, "Residue codes for concurrent error detection in artificial neural networks," *Proc. IJCNN'93*, Portland, OR, July 1993.
- [7] S. Bettola and V. Piuri, "The use of redundant binary representation for fault tolerant neural networks," *Proc. Int'l Conference on Neural Networks*, Orlando, FL, July 1994.
- [8] J. Hertz and A. Krogh, *Introduction to the Theory of Neural Computation*, Wesly Publishing Co., 1991.
- [9] C. Lehmann and F. Blayo, "A VLSI implementation of a generic systolic synaptic building block for neural networks," *Proc. Int'l IFIP Workshop on VLSI for Artificial Intelligence and Neural Networks*, St. Paul de Vence, France, 1990.
- [10] J. A. Abraham and W. K. Fuchs, "Fault and error models for VLSI," *IEEE Proceedings*, May 1986.
- [11] D. A. Reynolds and G. Metze, "Fault detection capabilities of alternating logic," *IEEE Trans. Comput.*, Vol. 27, pp. 1093-1098, 1978.
- [12] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," *IEEE Trans. Comput.*, Vol. 31, pp. 589-595, 1982.
- [13] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.
- [14] Y.-M. Hsu and E. E. Swartzlander, Jr., "Time redundant error correcting adders and multipliers," *Proc. 1992 IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Dallas, TX, pp. 247-256.