

Time Redundancy for Error Detecting 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

Concurrent error detection at an architectural level is often a basic requirement to achieve fault tolerance in neural networks for mission-critical applications. Time redundancy allows for concurrent error detection with low circuit complexity. In this paper, the use of alternating logic and complemented logic are analyzed as low-cost approaches to concurrent error detection. Different architectural approaches for the neural network design are considered to match the implementation constraints.

1. Introduction

The intrinsic parallelism in neural computation is effective and efficient for applications in several areas (e.g., signal and image processing, robotics, and real-time control) [1]. Due to the advances in the integration technologies, dedicated implementations are now affordable so that they can be widely used in the near future. Consistency and correctness of the neural network's outputs need to be continuously certified when they are used in mission-critical applications like aerospace environments and critical control systems.

In some neural paradigms (e.g., multi-layered feed-forward networks), an intrinsic correction capability can be introduced in the neural computation by applying suitable training procedures [2]. The presence of a limited number of faults affecting the behavior of the underlying digital architecture is automatically tolerated due to the distribution of information and computation in the whole structure and to the non-linearity of the computation. However, intrinsic error masking does not provide any information either on the correctness of the current output or on the location of faulty components. Intrinsic error correction may fail for some input sets and from a specific time on, but no signal is generated to prevent the use of the erroneous data generated by the network in critical applications. Furthermore, subsequent faults may be fatal for the intrinsic error masking capability. Thus the network may abruptly fail without any indication.

All the above implies that mission-critical applications cannot rely on the values generated by neural architectures where fault tolerance is achieved only through the learning procedure. In order to guarantee effective fault-tolerance, it is necessary to provide error detection at the

architecture level by applying other techniques. Error detection is essential to certify the correctness of the network outputs. Fault location is the second step required to support fast replacement of the faulty component within the neural network or performance degradation in order to extend the system availability. The use of the neural paradigms in real-time environments requires on-line techniques for error detection which are executed concurrently with the normal system operation. For location, off-line testing could be adopted but all components should be tested as at the end of production. Or when minimization of the diagnosis and repair time is required to increase the system availability, concurrent fault location should be adopted.

In the literature, efficient algorithm-based approaches have been proposed for neural computation [3][4], but they are either too application-specific or require high circuit complexity. The use of data coding [5]-[7] has also been successful for time-critical implementations, even if the circuit complexity is increased.

In this paper, time redundancy is used to achieve concurrent error detection at a low increase of circuit complexity in digital neural networks. Several approaches have been presented in the literature for time redundancy [8]-[11]. Basically, all of them repeat the same computation several times by using the system components in a different way for each repetition; then, results are compared to detect the presence of errors due to hardware faults.

Alternating logic was used in [12] for error detection. The nominal operation is performed with the nominal operands in the computing phase, while the one's complements of the operands are used during the checking phase. If the circuit has been properly designed, in the absence of faults, the result of the checking phase should be the one's complement of the result of the computing phase. This property is valid for ripple-carry adders and multipliers [13], while it can be generalized in any digital circuit by applying the technique given in [14].

A variation of this technique is complemented logic, in which nominal and checking operands and results are each the two's complement of the other. The use of the two's complements allows for simplifying the overall architecture, despite a slightly reduced error detection capability.

Feed-forward multi-layered neural networks are considered in this paper as an example, but the techniques can be adopted for many other neural paradigms. Section 2 introduces the general architectures that can be used to implement the neural paradigm. Section 3 presents alternating and complemented logics. Section 4 analyzes the architectures for concurrent error detection with such techniques and the architectural support for concurrent fault location.

2. Neural computation and the digital architectures

Several architectures ranging from dedicated structures to general-purpose neurocomputers have been presented in the literature to implement neural networks. In this paper the focus is on the digital realizations of dedicated architectures capable of performing only one class of neural computation. That is, the topology is fixed and only the interconnection weights can be changed.

The multi-layered feed-forward networks [1] are considered as an example. The proposed techniques can be applied to any kind of neural paradigms. The operation of neuron i is defined by $y_i = f_i(\sigma_i - \theta_i) = f_i(\sum_k w_{ik} x_k - \theta_i)$, x_k being the neuron's input from neuron k , w_{ik} the interconnection weight from neuron k to neuron i , y_i the neuron's output, θ_i the threshold, σ_i the weighted summation of the neuron's inputs, and f_i the non-linear activation function. No restrictions are imposed either on the non-linear function or on the values of the interconnection weights. For the sake of simplicity, it is assumed that operands (neuron inputs and weights) are fixed point n -bit two's complement numbers.

Neural network architectures presented in the literature can be basically classified into three types (Fig. 1): direct-mapped, pipelined, and time-multiplexed. Direct mapped implementation (Fig. 1a) of the neural network is a one-to-one mapping of neural operators onto digital components [15]. In each neuron, every arithmetic module M_k generates one product term of the input weighted summation from one neuron's input and the corresponding interconnection weight. This term is then added to the partial summation produced by the previous module. The new partial summation is then propagated to the subsequent module, until the last module feeds the complete weighted summation into the activation function circuit. Finally, the neuron's output is delivered to the receiving neurons.

To increase the system throughput, pipelining can be adopted (Fig. 1b). The basic operations within the individual neuron are cascaded and separated in time by latching the data between adjacent arithmetic modules in registers. Now the period of input presentation is reduced to the latency of a single pipeline stage.

To decrease the overall circuit complexity, time-multiplexing of some components can be exploited (Fig. 1c), despite an increased latency and a reduced throughput. For example, only one multiplier and one adder-accumulator can be used for each neuron. Weights are circularly rotated in a register file within each neuron. Input presentation is achieved by multiplexing the interconnection structure between neurons in adjacent layers. Only one data path is used between adjacent layers. Neuron outputs are stored in parallel in the output registers and then circularly shifted during each clock cycle to present all of them at the proper time to the receiving neurons.

3. Time-redundant techniques for concurrent error detection

The time-redundant techniques used in this paper are based on repeating the same computation with different input data, providing that the relationship between the nominal and the checking inputs also holds between the nominal and the checking outputs. The digital architecture is built so that any fault in the neurons modifies at least one of the outputs (either the nominal or the checking output). Error detection in the neural output can therefore be performed by comparing the normal and the checking outputs.

3.1 Alternating logic

In alternating logic [9, 12, 14], the checking relationship is one's complementation. The checking inputs are the one's complement of the nominal operands, while the checking outputs should be the one's complement of the nominal results. The neural architecture performs the neural operations on the nominal operands during the first phase (the computing phase) of operation to generate the nominal results. During the subsequent checking phase, the neural operations are performed on the complemented operands to generate the checking outputs. Then, the nominal results are complemented and compared to the checking results. An error is detected whenever they differ. Fault masking may occur if the error induced by the fault does not affect the current result. In this case, the fault remains latent and does not affect the neural outputs, until a suitable input data excite the error.

Alternating logic applies to digital architectures that are self-dual [12]. A multi-input single-output combinational circuit is said to be self-dual if and only if $\phi(\zeta) = \overline{\phi(\bar{\zeta})}$, ζ being the set of inputs, $\phi(\cdot)$ the output, $\bar{\zeta}$ the set of complemented inputs, and $\overline{\phi(\cdot)}$ the complemented output. This definition can be easily extended to multi-input multi-output combinational circuits.

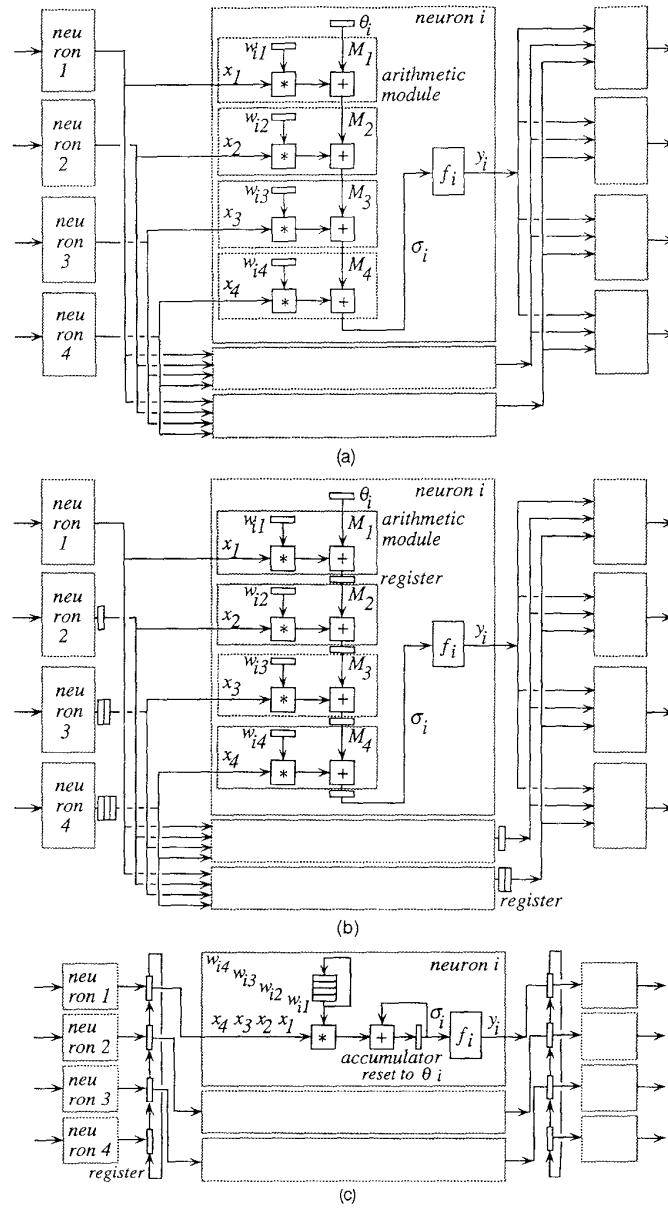


Fig. 1 - The neural architectures: (a) one-to-one mapped, (b) pipelined, (c) time-multiplexed.

Ripple-carry adders and multipliers can both be designed to be self-dual [13], with a very limited increase in circuit complexity. In [14], it was shown that any combinational circuit, $\varphi(\cdot)$, having N inputs can be transformed into a self-dual circuit, $\varphi_{sd}(\cdot)$, having $N+1$ inputs. Among all of the traditional activation functions for a neuron, only the step function with output values either 0 or 1 and discontinuity at the origin is self-dual. The alternating logic can also detect errors in the interconnection paths [12] since $\varphi(\zeta) = \zeta$ is a trivial function describing any interconnection path.

From the above discussion, it can be seen that neural architectures can always be designed to be self-dual. Hence, alternating logic can be applied to provide concurrent error detection.

3.2 Complemented logic

In complemented logic, two's complementation is used as the checking relationship. The checking inputs are the two's complement of the nominal operands, and the checking outputs should be the two's complement of the nominal results. It is obvious that in the two's complement case some bits (at most two) of the operands are treated in the checking phase the same as in the computing phase. Therefore, some faults are not detectable. However, by assuming a uniform probability distribution of inputs and a uniform probability distribution of faults at gate level, simulations have shown that the probability p of undetected errors is about 0.015. Since subsequent operations are not correlated when the operands are not correlated, the probability that the fault not being detected after k additions is p^k and quickly approaches 0. Having fixed the maximum acceptable undetected probability u for an application, one should discard at most the last k summations (where $p^k \leq u$) when an error is detected, in order to be sure of having accepted the previous results as correct with probability $1-u$. Complemented logic signals the presence of error so that results should be discarded from that time on, while the statistical evaluations mentioned in Section 1 (e.g., [2]) are not able to verify the correctness of the current output. Therefore, in applications where the probabilistic correctness certification is reasonable, complemented logic can be used to reduce the circuit complexity.

Achieving the two's complement relationship for the non-linear activation function is straightforward when this function is odd. For example, the following functions are widely used in neural literature and applications: linear function passing through the origin, step function with output values -1 or 1 and discontinuity at the origin, multi-step odd function with output values in $[-1, 1]$, and the sigmoid with output values in $[-1, 1]$.

Note that the threshold of the neuron can be considered as a special input and included in the weighted summation, so that its presence does not affect the two's-complement property of the activation function. The two's-complement relationship obviously holds in the interconnection paths since their operation is described by the identity function, as in the case of alternating logic.

Therefore, complemented logic can be used for concurrent error detection in neural architectures, even if the detection coverage is slightly less than 100%. The great advantage of complemented logic with respect to alternating logic is the lower increase of circuit complexity: adders and multipliers are in fact the traditional non-fault-tolerant structures in both phases, while no modification of the non-linear function is required in most of the practical cases.

4. Concurrent error detecting neural architectures

The techniques introduced in Section 3 for concurrent error detection can be applied to the three neural architectural classes discussed in Section 2.

4.1 Alternating logic

For the case of direct one-to-one mapping of the neural operators onto arithmetic components, the general scheme of the neuron's architecture with alternating logic is shown in Fig. 2. Since the weighted summation is a self-dual operation and the non-linear activation function can always be implemented by a self-dual circuit, the whole neuron is self-dual. Inverters and multiplexers are used at each primary input of the neuron to provide the one's complemented input and select the nominal or the complemented output according to the operating phase (Fig. 3). The general architectures of self-dual adders and multipliers [13] are very similar to their traditional counterparts. However, their circuit complexity is higher in order to accommodate the components required by both operation phases.

Protection of the activation function $f_i(\cdot)$ is dependent on the function itself. In the general case, a self-dual circuit can be designed as shown in [15]. This may require a circuit whose complexity is more than twice the nominal one since one of the circuits implements the nominal function, another implements the dual function, and a multiplexer selects the output. In the case of step function with output values either 0 or 1 and discontinuity at the origin, a single-bit data path connected to the most-significant bit of the weighted summation can be used. For some specific implementations, other dedicated error-detecting techniques can be adopted to protect the activation function generator separately from the linear part of the neural computation.

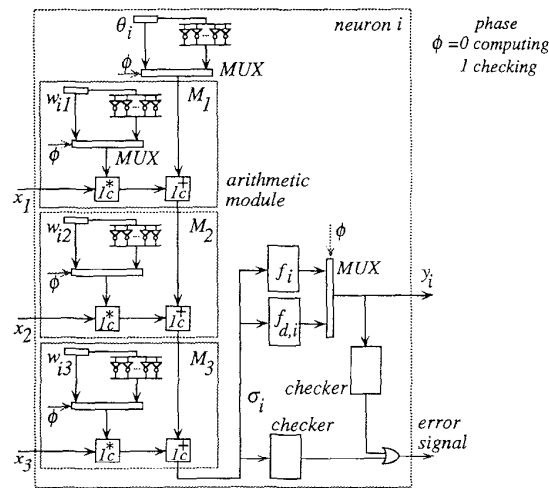


Fig. 2 - The neuron's architecture for the case of one-to-one mapping with alternating logic (where $f_i(\cdot)$ is not self-dual.)

Two checkers are used in Fig. 2 since the presence of the non-linear activation function may mask some errors in the linear part. The checker on the weighted summation can be avoided only if the activation function is linear. To check the weighted summation, the nominal value generated during the computing phase is held in a register and complemented, by the checking circuit shown in Fig. 4. Then, when the complemented value of the weighted summation has been generated in the checking phase, it is compared with the one's-complement of the nominal value. If they are not identical, the error is detected. The use of self-checking checkers or

duplication of the comparators can guarantee the proper operation of the checkers, otherwise the comparators should be considered hard-core. The checker at the output of the activation functions can be omitted to reduce the circuit complexity without affecting the error detection capabilities if the function is linear. This is because errors in the activation function are propagated to the following neurons and can be detected by the receiving neurons as they were errors in the circuits for the weighted summations themselves.

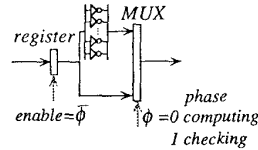


Fig. 3 - Generator for checking data at the primary inputs of the network for alternating logic.

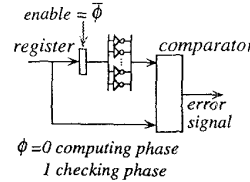


Fig. 4 - Checker for alternating logic.

By observing the error signals generated within each individual neurons, it is possible to locate the faulty neuron. If a checker is used at neuron's output, the identification of the faulty neuron is easy. It is the one in the layer nearest to the network's inputs among the neurons with active error signal. If checking is only performed on the weighted summation, the neuron with a fault in the linear part is located as in the previous case. On the other hand, if a fault induces an error in the activation function, such an error affects all the computation of weighted summations in the receiving neurons so that all of them have active error signals. In this case, only the layer containing the faulty neuron can be identified, not the faulty neuron itself. At the network level, the extraction of all error signals from the individual neurons to locate faults may not be feasible. In this case, a regular network can be designed for the error signal, as shown in Fig. 5. A triangularization procedure allows for identifying the faulty neuron by inspecting only the column and row error signals. The faulty neuron is the one in the leftmost column having active column error signal, and in the row with active row error signal.

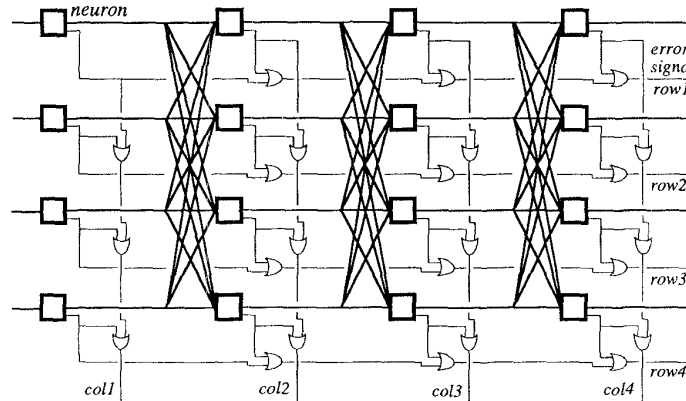


Fig. 5 - The error signal propagating network for locating the faulty neuron at the network level.

The direct-mapped architecture can be easily pipelined to increase the throughput. One register is introduced in each arithmetic module to hold the partial summation between adjacent modules, and registers are used at neurons' outputs to store the value that must be distributed to the receiving neurons.

The general structure of the time-multiplexed neural architecture with alternating logic is shown in Fig. 6. The core is basically identical to the arithmetic module for the two previous architectures. The register file is twice as long as the nominal one. The nominal weights are stored in the first N positions, while their one's complements are in the subsequent N positions. No multiplexer is required here. The neuron's threshold θ_i can be incorporated in the weighted summation as the initial value of the accumulator register at the beginning of each phase. The nominal threshold is used for the computing phase, while its one's complement is applied for the checking phase. Result checking and the protection of the activation function can be achieved as the direct-mapped architecture. At the network level, the same error signal propagating network presented in Fig. 5 can also be used to extract the error signals and locate the faulty neuron.

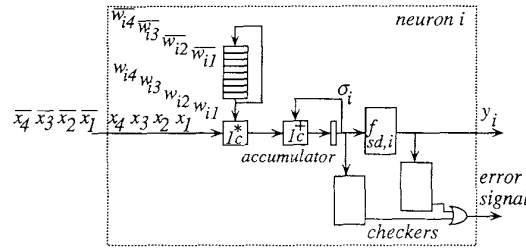


Fig. 6 - The neuron's architecture for the case of the time-multiplexed structure with alternating logic (where $f_i(\cdot)$ is self-dual.)

4.2 Complemented logic

The application of the complemented logic leads to architectures similar to the ones for alternating logic. Consider first the case of the direct-mapped approach. The general structure is shown in Fig. 7. The circuit generating the two's-complemented inputs for the checking phase is shown in Fig. 8. The input register holds the current network input, while the ripple-carry chain of half adders generates the two's complement of the inputs. A multiplexer selects the appropriate input according to the current operation phase. Concerning the weights, a possible solution could be using the same circuit for the network inputs. A better approach is to duplicate the weight registers in every neuron. One set of registers contains the nominal weights for the computing phase, the other holds the two's-complemented weights for the checking phase. Multiplexers select the appropriate values according to the operation phase.

The arithmetic units in the arithmetic modules are exactly the same as those of the nominal architecture. The threshold θ_i or its two's complement is applied at the summation input of the arithmetic module M_i in each neuron, as in the case of alternating logic, according to the current operation phase. As already stated in Section 3.2, protection of the activation function is simple if it is odd. Otherwise, the approach based on self-complemented general functions can be adopted.

Checking is performed by the circuit shown in Fig. 9 on weighted summations and, possibly, on the neuron outputs, in a way similar to the one discussed for alternating logic. Since the value stored in the register of the checker should be the two's complement of the value generated

during the checking phase, error detection can also be performed simply by adding these data. Their sum should be zero if no error occurred. In this case, the checker consists of an adder and an OR gate that generates the error signal from all the sum bits of the adder. High-speed adders can be used to reduce the latency.

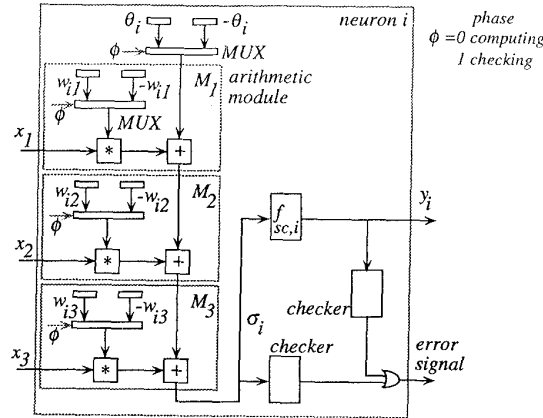


Fig. 7 - The neuron's architecture for the case of one-to-one mapping with complemented logic (where $f_i(\cdot)$ is an odd function.)

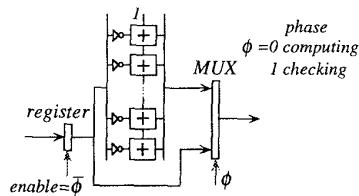


Fig. 8 - Generator for checking data at the primary inputs of the network for complemented logic.

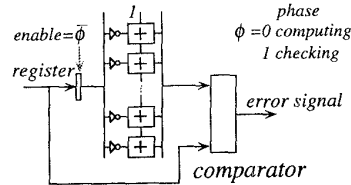


Fig. 9 - Checker for complemented logic.

Locating the faulty neuron can be performed as in alternating logic. At the network level, the error signal network of Fig. 5 and the triangularization technique discussed in Section 4.1 can also be applied to simplify error signal extraction and fault location.

The pipelined architecture for complemented logic can be derived from the direct-mapped structure by introducing the suitable registers for data latching between adjacent pipeline stages. The resulting architecture is similar to the one discussed above. Finally, the time-multiplexed architecture for complemented logic can be derived. The resulting structure is shown in Fig. 10.

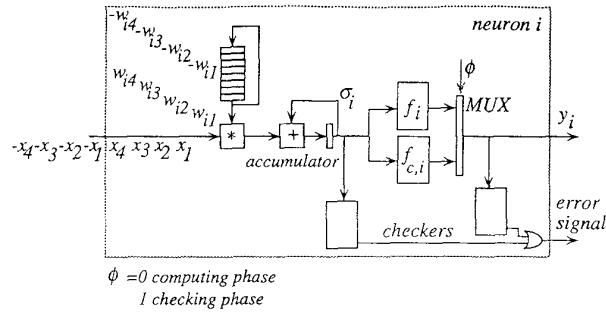


Fig. 10 - The neuron's architecture for the case of the time-multiplexed structure with complemented logic (where $f_i(\cdot)$ is not self-complemented.)

5. Concluding remarks

Two approaches for designing dedicated neural architectures with concurrent error detection have been introduced using alternating logic and complemented logic. The first technique guarantees complete protection from any single error with a low increase in circuit complexity. In the direct-mapped architecture, for a moderate number of neurons and for an efficient implementation of a simple activation function, the circuit complexity increase is about 20% and this is practically independent of the number of bits in the operands and the number of arithmetic modules. Latency in the architecture without pipelining is about twice that of the nominal structure, and the throughput is one half of the nominal one. When pipelining is adopted, the throughput is higher but is still half of the pipelined nominal architecture since one half of the computation capabilities must be spent to generate the checking data. In the time-multiplexed structure, this technique is quite expensive in terms of circuit complexity due to the intrinsic compactness of the nominal architecture.

The second technique using complemented logic allows for circuit complexity lower than that of the alternating logic, despite a slight reduction of the error detection coverage. In the direct-mapped structure, the circuit complexity increase is sensitive to the number of bits in the operands. It is about 17% for $n=6$, 12% for $n=8$, 9% for $n=10$, and 8% for $n=12$. By assuming a uniform distribution of neural inputs, the probability that an erroneous result is not detected is about 0.015.

References

1. J. Hertz and A. Krogh, *Introduction to the Theory of Neural Computation*, Wesley Publishing Co., 1991.
2. M. Stevenson, R. Winter, and B. Widrow, "Sensitivity of feedforward neural networks to weight errors," *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, March 1990.
3. V. Piuri, "An algorithmic approach to concurrent error detection in artificial neural networks," *Proc. IEEE Int'l Workshop on VLSI Signal Processing*, Napa, CA, October 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*, San 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. 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.
9. B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.
10. J. Li and E. E. Swartzlander, Jr., "Concurrent error detection in ALUs by recomputing with rotated operands," *Proc. 1992 IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Dallas, TX, pp. 109-116.
11. 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.
12. D. A. Reynolds and G. Metze, "Fault detection capabilities of alternating logic," *IEEE Trans. Comput.*, Vol. 27, pp. 1093-1098, 1978.
13. K. Takeda and Y. Tohma, "Logic design of fault-tolerant arithmetic units based on the data complementation strategy," *Proc. 1980 IEEE Int'l Fault Tolerant Computing Symposium (FTCS-10)*, pp. 348-350.
14. Z. Kohavi, *Switching and Finite State Automata Theory*, McGraw-Hill, New York, 1978.
15. 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.