FAULT TOLERANT NEURAL NETWORKS WITH HYBRID REDUNDANCY

Lon-Chan Chu and Benjamin W. Wah

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 West Springfield Avenue
Urbana, IL 61801
chu%aquinas@uxc.cso.uiuc.edu

ABSTRACT

In this paper, a fault-tolerant neural network with hybrid redundancy is proposed and analyzed. A hybrid redundancy is a combination of spatial redundancy, temporal redundancy, and coding. It is based on the homogeneity of both structures and operations of neurons. By storing multiple sets of weights in a processor and by recomputing the outputs of neurons with multiple processors, faults in the processors can be detected and corrected. This architecture can highly increase the reliability of a neural network so that a fairly large number of faulty neurons can be detected and that the outputs of these faulty neurons can be recovered. The redundancy of this architecture is fairly low if only certain critical neurons, such as output neurons, are implemented with this technique.

KEYWORDS AND PHRASES. Error correcting codes, hardware implementation, hybrid redundancy, multi-layer artificial neural networks, spatial redundancy, temporal redundancy.

1. INTRODUCTION

Neural networks have strong potentials for applications in robotics, signal processing, pattern classification, and combinatoric optimization [2,4,5]. Although neural networks are robust to failures, neurons affecting outputs that control critical devices must be reliable. Failures of these neurons may cause incorrect signals to be sent to these critical devices.

Two methods can be applied to increase the reliability of neural networks. First, possible failures can be accounted for in the training process, so the network can recover from these failures when they occur. Second, as suggested by Moore [6], the neurons can be made fault tolerant, so that failures can be recovered without affecting the outputs of neurons.

We approach the problem of increasing the reliability of neural networks by designing fault-tolerant neurons. We present in this paper a powerful but simple hybrid redundancy method, which can be applied on a *well-trained* multi-layer neural network. In such networks, the neurons can be classified into a limited number of isomorphic sets in which neurons in a set have identical inputs (or activations from neurons in other sets). By replicating the weights used by one neuron in a set to other neurons in the same set, other neurons in the set can be used to recompute the output of the given neuron. A similar idea of recomputation has been proposed for bit-sliced ALUs [7] and iterative logic arrays [1]. We show in this paper that the resulting design is very robust, even when a considerable number of neurons are faulty.

The faults in a system may be transient, intermittent, or permanent [3]. Traditional fault-tolerance strategies include spatial redundancy, temporal redundancy, and coding [8]. A typical technique of spatial redundancy is the triple module redundancy (TMR) which is useful for coping with transient, intermittent, and permanent faults. The hybrid redundancy model we propose is more reliable and less costly than the TMR, which fails when two-out-of-

This research was supported partly by National Science Foundation Grant MIP 88-10548 and by National Aeronautics and Space Administration Grant NAG 1-613.

International Joint Conference on Neural Networks, 1990.

three redundant neurons fail. Our proposed method uses spatial as well as temporal redundancy to recompute the outputs using a number of neurons receiving identical inputs. It can cope with transient, intermittent, and permanent faults. It improves traditional methods based on temporal redundancy, which recompute outputs in time and are generally useful for filtering out transient faults. Our proposed method also incorporates error correcting codes in correcting errors in circuits that cannot be covered by spatial and temporal redundancies.

This paper is organized as follows. Sections 2 and 3 define the hybrid redundancy model and the model of neural networks, respectively. Section 4 describes the mechanism for constructing a neural network based on the hybrid redundancy model as a building block. Sections 5 and 6 show the reliability and overhead of our method. Conclusions are drawn in Section 7.

2. M-WAY HYBRID REDUNDANCY MODEL

In this section, we present an m-way hybrid redundancy model based on spatial and temporal redundancies. The model is defined formally as **HR** = <**P**, B, DA, m>, where **P** is a set of n *synchronous* processing elements (PE), B is a *broadcast bus* used by the external host to broadcast inputs to all PEs in **P**, DA is a *decision automata* for error detection and recovery, and m is the degree of redundancy. The architecture is shown in Figure 2.1. The DA is a ring of n decision cells (DC) connected in the form of a ring. All PEs are synchronous in the sense that they receive inputs, produce results, and sends outputs at the same time. Each PE is associated with a DC, which produces the recovered outputs and reports the fault status of its corresponding PE. We assume that the outputs produced by the PEs are continuous and bounded and that the bus is fault-free. The first assumption implies that each output value can be represented in a fixed number of binary bits. Note that our design also applies to cases in which the output values of neurons are binary or discrete. The assumption on the reliability of the bus is reasonable because the hardware complexity of the bus is much smaller than that of the rest of the system.

A special case of the hybrid redundancy model is the non-redundant model **NR=<P,B>**. This has the same architecture as that of **HR** except that it has one storage bank in each PE and no decision cells.

The architecture of $PE_i \in \mathbf{P}$ consists of an input buffer, m+1 storage banks (SB_1, SB_1, ..., SB_m) of equal size, an arithmetic and logic unit (ALU), a register file, an error-correcting-code (ECC) encoder, an ECC decoder, and an output buffer (see Figure 2.2). The SBs are used to keep operands, such as the connection weights of a neural network. To achieve the m-way spatial redundancy, $SB_{i,k}$ of PE_i has the same content as $SB_{(i+k) \bmod n,\, 0}$, where

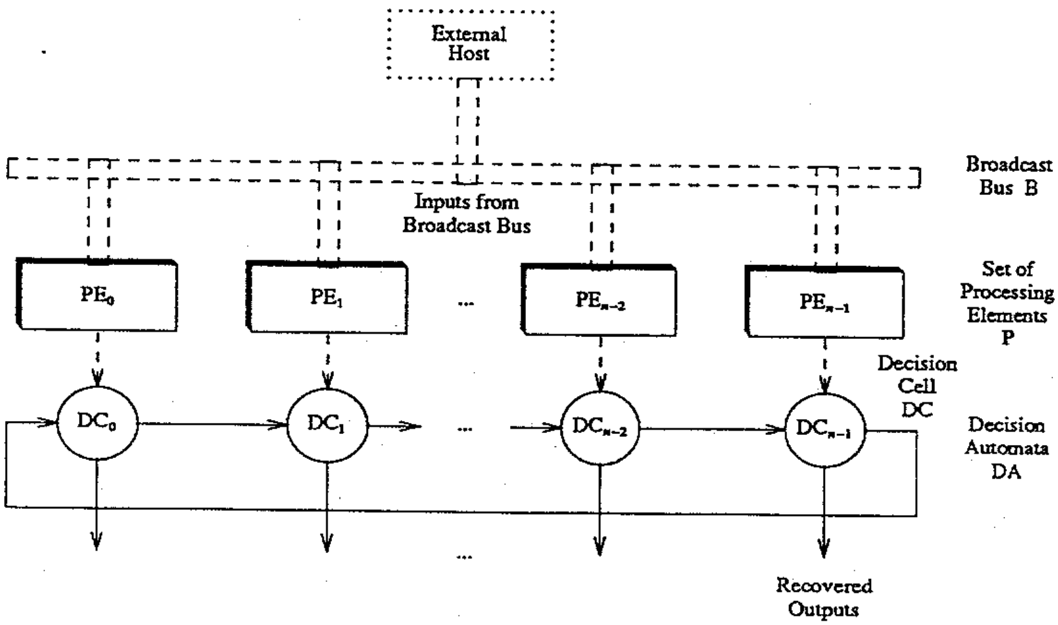

Figure 2.1. Architecture of the m-way hybrid redundancy model.

$0 \le i \le n-1$ and $1 \le k \le m$. The register file keeps temporary operands and intermediate results. The control unit executes and issues all necessary operations. The ECC encoder generates single-error-correction, double-error-detection (SECDED) codes and append them to output codes. The ECC decoder uses the SECDED code to correct a single error or detect double errors in the inputs. Both can be implemented with simple combinatorial logic gates. The PEs corresponding to input and output neurons have no ECC decoders and encoders. Error correction and detection are implemented in the PEs rather than the DAs in order to minimize the logic circuits in the latter.

A PE operates in major cycles, each with $m+1$ minor cycles. PE_i receives inputs from the broadcast bus at the same time as other processors. It then repeats a sequence of operations for $m+1$ minor cycles. In minor cycle k, $0 \le k \le m$, it compute its result based on the values in $SB_{i,k}$ and output the results to DC_i. The sequence of instructions carried out in each cycle are identical for all processors and all cycles (using different inputs). The major cycle is then repeated for another set of inputs from the host.

Redundancy in space and time are used in this model. The m-way spatial redundancy is achieved by assigning $SB_{i,k}$ of PE_i to have the same content as $SB_{(i+k) \bmod n, 0}$, where $0 \le i \le n-1$ and $1 \le k \le m$. The m-way temporal redundancy is accomplished by the m extra cycles in each PE. Since (a) all PEs receive identical inputs in a major cycle, (b) $SB_{i,k}$ of PE_i has the same content as $SB_{(i+k) \bmod n, 0}$, (c) operations carried out in all minor cycles are identical, and (d) minor cycle k in PE_i uses inputs from $SB_{i,k}$ and the host, we can conclude that the outputs generated by PE_i in minor cycle k is the same as the outputs generated by $PE_{(i-j) \bmod n}$ in minor cycle $k-j$, where $0 \le j \le k$, assuming all PEs are fault-free. Redundancy in the form of error correcting codes are implemented in the PEs in order to detect failures in the DCs. A single error in the generation of the recovered code in a DC can be corrected using the SECDED code.

The DA is a ring of n DCs, each of which is responsible for correcting the output of the corresponding PE. Suppose PE_i generates its output at time t. This output is redundantly generated by PE_{i-j}, $1 \le j \le m$, at time $t+j$. These $m+1$ redundant outputs are shifted sequentially into DC_i, which computes the majority of the $m+1$ codes.

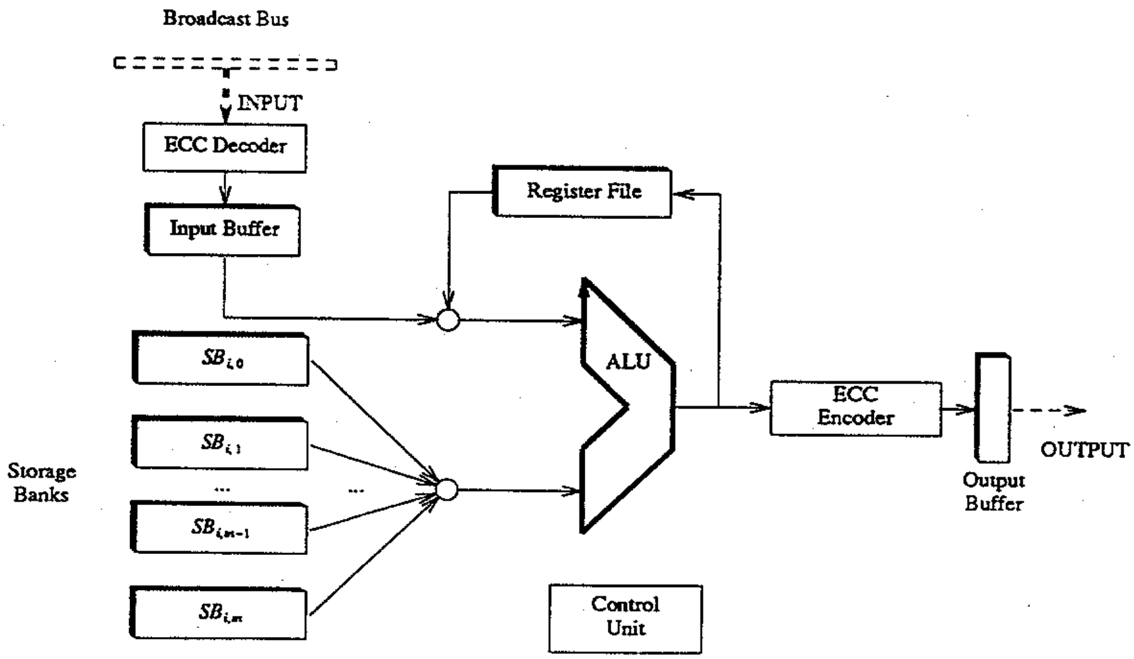

Figure 2.2. Architecture of a PE.

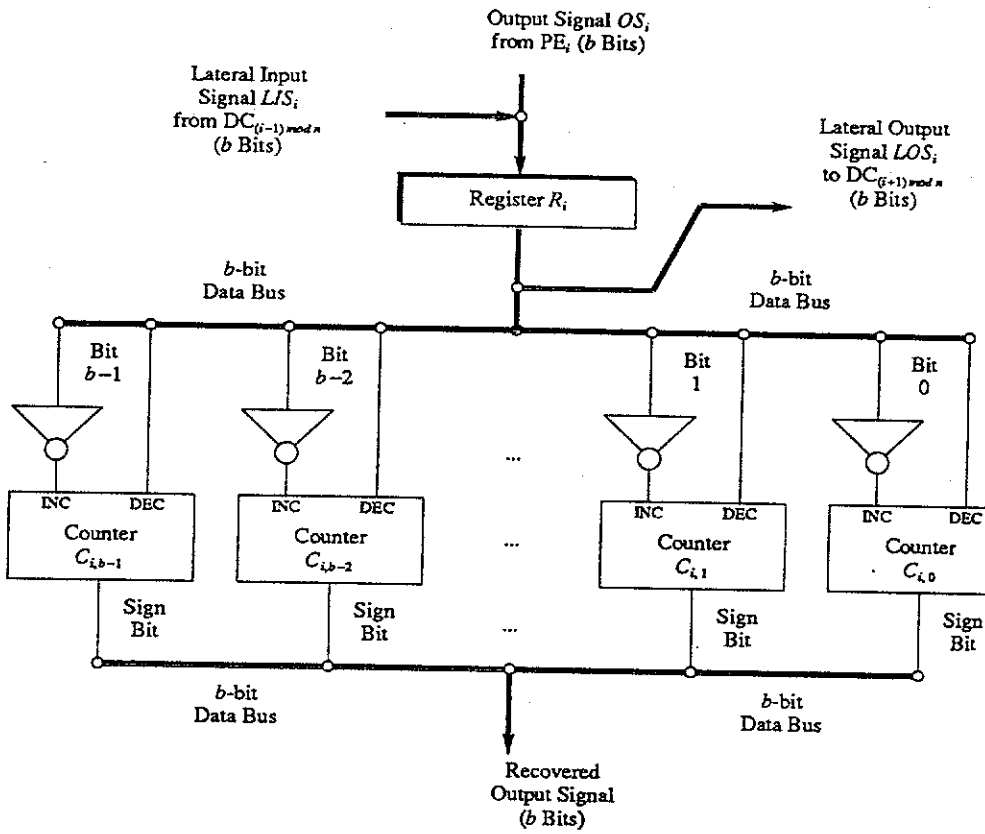

Figure 2.3. Architecture of DC_i.

The architecture of DC_i is shown in Figure 2.3. Its left and right neighboring cells are $DC_{(i-1) mod n}$ and $DC_{(i+1) mod n}$, respectively. DC_i receives its inputs from PE_i in the first minor cycle of each major cycle and from $DC_{(i-1) mod n}$ (as signal LIS_i) in subsequent minor cycles. Without loss of generality, let the inputs received by DC_i be represented in b bits, which include the data and the SECDED code. DC_i has a b-bit register R_i and b up-down binary counters $C_{i,0}, C_{i,1}, ..., C_{i,b-1}$. R_i is used for holding inputs from PE_i or $DC_{(i-1) mod n}$, which are connected in the form of a wired-OR. The following operations are performed in DC_i in a major cycle, which can be carried out concurrently with operations in PE_i.

(1) Reset the b counters in DC_i.

(2) Repeat Steps 3 and 4 synchronously with other DCs for $m+1$ minor cycles.

(3) In minor cycle j, $0 \le j \le m$, wait until the output of PE_i is ready and send it sequentially through the registers to R_{i+j}, the register in DC_{i+j}, for $0 \le i \le n-1$. This requires j shifts of the data through the registers.

(4) Using the value stored in register R_i, accumulate bit $R_{i,x}$, $0 \le x \le b$, of R_i into counter x in the following way. If $R_{i,x}$ is 1, then decrement $C_{i,x}$, otherwise increment $C_{i,x}$.

(5) The b sign bits of counters $C_{i,b-1}, ..., C_{i,0}$ form the recovered output of PE_i.

The above steps correctly identify a majority of the $m+1$ redundant codes according to the following lemma.

Lemma 2.1. Given $m+1$ codes generated redundantly, m being even, and assuming $m/2$ or less incorrect codes, the steps carried out in DC_i correctly identify a majority of the $m+1$ codes.

Proof. Since there are $m/2$ or less incorrect codes, $m/2+1$ codes must be identical and forms a majority. This implies that bit position j of the majority code can be computed by finding the majority of the $m+1$ bits in bit position j of the $m+1$ codes. The up/down counters shown in Figure 2.3 achieve this purpose. At the end of a major cycle, the sign bit of $C_{i,j}$ correctly identifies the majority of bits seen by this counter. □

The design of DC_i is efficient, since finding the majority of $m+1$ (m even) bits requires a counter with a minimum $\lceil \log_2(m+3) \rceil$ positions. b counters are needed, one for each bit position. In our design in Figure 2.3, we assume that counters with $\lceil \log_2(2m+3) \rceil$ bits are used. Although an additional flip/flop may be required for each counter, the combinatorial logic controlling the counter is simpler than a design with $\lceil \log_2(m+3) \rceil$ bits.

The proposed design of decision cell requires the output of PE_i to be shifted by a variable number of positions, depending on the number of minor cycles that has already been carried out in a major cycle. This varying number of shifts is necessary in order to bring the redundant output generated for PE_i at a distance j away to be available at DC_i, where j is the number of minor cycles that have already been carried out. These shifts do not become a bottleneck because the complexity of operations in a PE is generally much more complex than the operations implemented in a decision cell.

Note that the b bits recovered in each DC includes the SECDED code. This means that the PE receiving the recovered output can correct errors due to the failure of a single counter and detects errors due to the failures of two counters in a DC.

An example illustrating the operations in DC_i is shown in Section 4.

3. MODEL OF NEURAL NETWORKS

A neural network is characterized by a set **N** of N neurons, N weight vectors **W**, and the interconnection pattern. Neuron i is associated with weight vector $\mathbf{W}_i$. The interconnection pattern defines the data dependence of neural-network operations. According to the interconnection pattern, neuron i is associated with a set of predecessor neurons and a set of successor neurons. A predecessor of neuron i sends its output to neuron i in the production phase. A successor of neuron i receives the output of neuron i. Let $\mathbf{A}_i$ and $\mathbf{B}_i$ be the sets of predecessor and successor neurons of neuron i. Note that $\mathbf{W}_i$ has cardinality $|\mathbf{A}_i|$.

Neurons i and j are *isomorphic* if and only if both $\mathbf{A}_i=\mathbf{A}_j$ and $\mathbf{B}_i=\mathbf{B}_j$. A set of mutually isomorphic neurons is called an isomorphic set. An isomorphic set for a neuron is maximal if it is the largest of all possible isomorphic sets including this neuron. A multi-layer neural network can be characterized by isomorphic sets. For example, for a 3-layer neural network with full interconnection between adjacent layers, there are three maximal isomorphic sets since each layer corresponds to a maximal isomorphic set. It is assumed that only maximal isomorphic sets are used in the following discussion.

The representation of a neural network can be simplified using isomorphic sets. Note that if two isomorphic sets are connected, then every neuron in one set is connected to all neurons in the second set. A predecessor isomorphic set of isomorphic set **I** sends the outputs of its neurons to all neurons in **I**. Likewise, a successor isomorphic set of **I** receives the outputs of all neurons in **I**. The predecessor and successor isomorphic sets of **I** are denoted by $\mathbf{A_I}$ and $\mathbf{B_I}$, respectively. Let Θ be the set of all isomorphic sets in the neural network.

A neural network is *well-trained* if all its weight vectors are fixed and do not change during neural-network operations. It is assumed that all neural networks discussed in this paper are well-trained.

4. FAULT TOLERANT NEURAL NETWORK

A mechanism for constructing a reliable neural network using the hybrid redundancy model is described in this section. This design is based on the homogeneity of neurons in an isomorphic set. The steps for construction are described as follows.

(1) Determine all maximal isomorphic sets in the neural network. Denote the set of all maximal isomorphic sets as Θ. For isomorphic set $\mathbf{I} \in \Theta$, label all neurons uniquely by a number between 0 and $|\mathbf{I}|-1$.

(2) Select a set of isomorphic sets to be implemented using the hybrid redundancy model. This set, denoted by Θ_C, is called the *critical kernel*. An example of a critical isomorphic set is an isomorphic set of output

neurons. All isomorphic sets not selected are implemented without redundancy.

(3) Associate isomorphic set $\mathbf{I}$ in Θ_C with a redundancy model $\mathbf{HR_I}$=<$\mathbf{P_I}$,$\mathrm{B_I}$,$\mathrm{DA_I}$,$m_\mathbf{I}$> such that $n_\mathbf{I}$, the number of PEs in $\mathbf{P_I}$, is equal to the number of neurons in $\mathbf{I}$, *i.e.* $n_\mathbf{I} = |\mathbf{I}|$. Define a *bijection* (one-to-one) mapping $\pi_\mathbf{I}$ between the neurons in $\mathbf{I}$ and the PEs in $\mathbf{P_I}$ such that for $\mathrm{PE}_i \in \mathbf{P_I}$, $\pi_\mathbf{I}(i)$ represents the corresponding neuron (i mod $n_\mathbf{I}$) in isomorphic set $\mathbf{I}$. That is, PE_i emulates neuron $\pi_\mathbf{I}(i)$. For every $\mathrm{PE}_i \in \mathbf{P_I}$, store $\mathbf{W}_{\pi_\mathbf{I}(i)}$ in storage bank $SB_{i,0}$ and $\mathrm{W}_{\pi_\mathbf{I}(i+k)}$ in storage bank $SB_{i,k}$ for k=1,...,$m_\mathbf{I}$. Note that $\mathbf{W}_{\pi_\mathbf{I}(i)}$ is the weight vector of neuron *labeled* $\pi_\mathbf{I}(i)$ in the same isomorphic set. Define the external hosts of $\mathbf{HR_I}$ as all PEs corresponding to the predecessor isomorphic sets of $\mathbf{I}$.

(4) Associate every isomorphic set $\mathbf{I}$ not in Θ_C with a non-redundant model $\mathbf{NR_I}$=<$\mathbf{P_I}$,$\mathrm{B_I}$>. For each PE_i, store $\mathbf{W}_{\pi_\mathbf{I}(i)}$ in the single storage bank. The external hosts are defined in the same way as those in the previous step.

The resulting architecture is a network of hybrid redundancy models **HRs** and non-redundant models **NRs**. The following example illustrates the construction method.

Example 4.1. Consider a 3-layer, fully-connected neural network shown in Figure 4.1a. Layers 1, 2, and 3 have 100, 200, and 14 neurons, respectively. To simplify the discussion and without loss of generality, all output values are assumed binary. Figure 4.1b shows the set Θ of isomorphic sets. For the neural network in Figure 4.1a with three layers, there are three maximal isomorphic sets $\mathbf{I}_1$, $\mathbf{I}_2$, and $\mathbf{I}_3$. Neurons in each isomorphic set are labeled accordingly.

The second step is to select the critical kernel Θ_C. As an illustration, assume $\Theta_C = \{ \mathbf{I}_3 \}$; that is, only the output layer (isomorphic set $\mathbf{I}_3$) is implemented using the hybrid redundancy model.

The third step is to associate every isomorphic set in the critical kernel Θ_C with the hybrid redundancy model. Assuming a redundancy degree of 6,

$$\mathbf{HR}_{\mathbf{I}_3} = < \mathbf{P}_{\mathbf{I}_3} = \{\mathrm{PE}_0, \cdots, \mathrm{PE}_{13}\}, \mathrm{B}_{\mathbf{I}_3}, \mathrm{DA}_{\mathbf{I}_3} = \{\mathrm{DC}_0, \cdots, \mathrm{DC}_{13}\}, m_{\mathbf{I}_3} = 6 > \tag{4.1}$$

$\mathrm{PE}_i \in \mathbf{P}_{\mathbf{I}_3}$ emulates neuron $i \in \mathbf{I}_3$. By definition, $\pi_{\mathbf{I}_3}(i) = i$ mod 14. For $\mathrm{PE}_i \in \mathbf{P}_{\mathbf{I}_3}$, $SB_{i,0}$ stores $\mathbf{W}_i$, and $SB_{i,j}$ stores $\mathbf{W}_{\pi_{\mathbf{I}_3}(i+j)}$, $1 \le j \le 6$, where $\mathbf{W}_{\pi_{\mathbf{I}_3}(i+j)}$ is the weight vectors of neuron $\pi_{\mathbf{I}_3}(i+j)$ in $\mathbf{I}_3$. These weights are shown in Table 4.1a. The external hosts are those PEs in $\mathbf{NR}_{\mathbf{I}_2}$.

The fourth step is to associate the isomorphic sets not in Θ_C, namely, $\mathbf{I}_1$ and $\mathbf{I}_2$, with $\mathbf{NR}_{\mathbf{I}_1}$ and $\mathbf{NR}_{\mathbf{I}_2}$. Since these are not interesting for our discussion, they will not be presented here. Note that the speed of $\mathbf{NR}_{\mathbf{I}_1}$ and $\mathbf{NR}_{\mathbf{I}_2}$ may have to be slowed down to match the speed of $\mathbf{HR}_{\mathbf{I}_3}$.

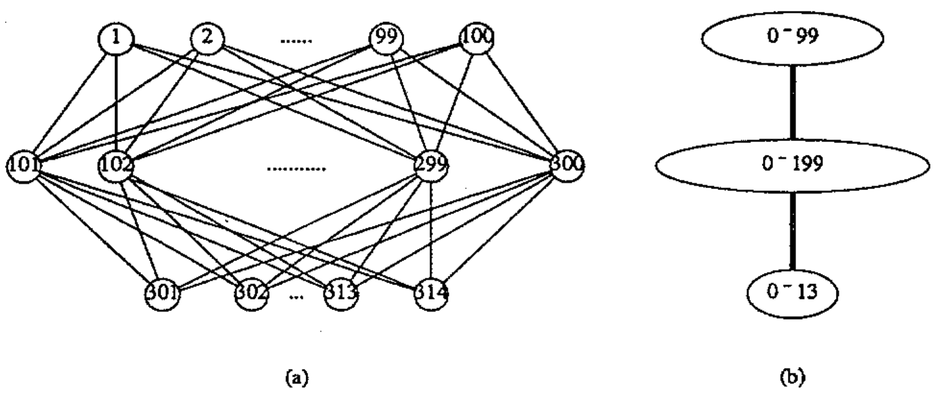

Figure 4.1. A 3-layer, fully-connected neural network represented as (a) neurons and (b) isomorphic sets.

Table 4.1a. Contents of Storage Banks in all PEs.

PE_i	Storage Banks 0	1	2	3	4	5	6	PE_i	Storage Banks 0	1	2	3	4	5	6
PE_0	W_0	W_1	W_2	W_3	W_4	W_5	W_6	PE_7	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
PE_1	W_1	W_2	W_3	W_4	W_5	W_6	W_7	PE_8	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_0
PE_2	W_2	W_3	W_4	W_5	W_6	W_7	W_8	PE_9	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_0	W_1
PE_3	W_3	W_4	W_5	W_6	W_7	W_8	W_9	PE_{10}	W_{10}	W_{11}	W_{12}	W_{13}	W_0	W_1	W_2
PE_4	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	PE_{11}	W_{11}	W_{12}	W_{13}	W_0	W_1	W_2	W_3
PE_5	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	PE_{12}	W_{12}	W_{13}	W_0	W_1	W_2	W_3	W_4
PE_6	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	PE_{13}	W_{13}	W_0	W_1	W_2	W_3	W_4	W_5

Table 4.1b. Outputs produced by all PEs.

Minor Cycle k	Outputs of PEs o_0	o_1	o_2^*	o_3	o_4	o_5	o_6^*	o_7	o_8	o_9	o_{10}	o_{11}	o_{12}	o_{13}
0	0	1	0	0	1	1	1	1	1	0	1	1	0	1
1	1	1	0	1	1	0	1	1	0	1	1	0	1	0
2	1	0	0	1	0	1	1	0	1	1	0	1	0	1
3	0	1	0	0	1	1	1	1	1	0	1	0	1	1
4	1	1	0	1	1	0	1	1	0	1	0	1	1	0
5	1	0	0	1	0	1	1	0	1	0	1	1	0	1
6	0	1	0	0	1	1	1	1	0	1	1	0	1	1

Table 4.1c. Signals fed into the counters in all DCs.

Minor Cycle k	Outputs $o_{\pi_{I_3}(i-k)}$ fed into the Up/Down Counter of DC_i 0	1	2*	3	4	5	6*	7	8	9	10	11	12	13
0	0	1	0	0	1	1	1	1	1	0	1	1	0	1
1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
2	0	1	1	0	0	1	0	1	1	0	1	1	0	1
3	0	1	1	0	1	0	0	1	1	1	1	1	0	1
4	0	1	1	0	1	1	0	1	1	0	1	1	0	1
5	0	1	1	0	1	1	0	0	1	0	1	1	0	1
6	0	1	1	0	1	1	0	1	1	0	1	1	0	1
Sign Bit of Counter	0	1	1	0	1	1	0	1	1	0	1	1	0	1
Error Status	0	0	1	0	0	0	1	0	0	0	0	0	0	0
Recovered Output	0	1	1	0	1	1	0	1	1	0	1	1	0	1

To simplify the discussion, only the operations in $\mathbf{HR}_{I_3}$ are described. Here, only one production phase is illustrated, since all production phases are similar. Assume the correct output is

$$o_0 \cdots o_{14} = 0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,1 \tag{4.2}$$

Without loss of generality, assume that PE_2 and PE_6 are faulty and that their outputs are stuck at 0 and 1, respectively. Table 4.1b shows all $m_{I_3}+1$ outputs produced. No error-correcting codes are shown in the table. Note that O_2^* and O_6^* represent stuck-at faults. Table 4.1c shows all signals fed into the up/down counter in DC_i. Note that if the entry $e_{k,i}$ in Table 4.2c is inconsistent with the majority of the column, then $PE_{\pi_{I_3}(i-k)}$ is faulty.

5. FAULT TOLERANCE ANALYSIS

In this section, we present the conditions on the redundancy degree necessary for reliable operations. Consider an isomorphic set I and its corresponding $\mathbf{HR}_\mathrm{I} = <\mathrm{P}_\mathrm{I}, \mathrm{B}_\mathrm{I}, \mathrm{DA}_\mathrm{I}, m_\mathrm{I}>$. The outputs of neuron i is computed at different PEs at different times. Let $N_n(i)$ be the number of outputs of neuron i computed correctly in various PEs, and $N_f(i)$ be the number of incorrect outputs computed. Define $N_m(i)$ as the *majority difference* between $N_n(i)$ and $N_f(i)$ of neuron i; that is,

$$N_m(i) = N_n(i) - N_f(i) \tag{5.1}$$

Note that $N_n(i)$ and $N_f(i)$ satisfies the following relation, given that m_I is the degree of redundancy.

$$N_n(i) + N_f(i) = m_\mathrm{I} + 1 \tag{5.2}$$

Combining Eq's (5.1) and (5.2) yields

$$N_m(i) = m_\mathrm{I} + 1 - 2N_f(i). \tag{5.3}$$

Faults in the outputs of neuron i can be recovered if there is a majority in the correct outputs generated for neuron i. This implies that the majority difference $N_m(i)$ should be greater than zero. We consider two cases below. Lemma 6.1 shows the necessary condition for recovering the correct output of neuron i when $m_\mathrm{I} < n_\mathrm{I}$. Lemma 6.2 shows the sufficient condition when $m_\mathrm{I} \geq n_\mathrm{I}$.

Lemma 6.1. Consider an isomorphic set **I** with n_I neurons and its $\mathbf{HR}_\mathrm{I}$ with n_f faulty PEs. If $m_\mathrm{I} < n_\mathrm{I}$, then the necessary condition for recovering the outputs of all neurons in **I** is

$$2n_f \leq m_\mathrm{I} \tag{5.4}$$

Proof. Let δ be a non-negative integer such that $m_\mathrm{I} = 2n_f + \delta$. In the worst case, $N_f(i) = n_f$. The majority difference can be derived by using Eq. (5.3).

$$N_m(i) = m_\mathrm{I} + 1 - 2n_f = (2n_f + \delta) + 1 - 2n_f = \delta + 1 > 0 \tag{5.5}$$

Since the majority difference is greater than zero, the outputs of PE_i can be recovered correctly. Hence, $\mathbf{HR}_\mathrm{I}$ is safe. □

Lemma 6.2. Consider an isomorphic set I with n_I neurons and its $\mathbf{HR}_\mathrm{I}$ with n_f faulty PEs. If $m_\mathrm{I} \geq n_\mathrm{I}$, then the sufficient condition for $\mathbf{HR}_\mathrm{I}$ to detect all faulty PEs and recover the correct outputs is

$$3n_f < n_\mathrm{I} \tag{5.6}$$

Proof. Let δ be a positive integer such that $n_\mathrm{I} = 3n_f + \delta$. There exist a positive integer p and a non-negative integer $q < n_\mathrm{I}$ such that $m_\mathrm{I} = p\, n_\mathrm{I} + q$. In the worst case, $N_f(i) = p\, n_f + q'$, where $q' = min(q, n_f)$. The majority difference can be derived by using Eq. (5.3).

$$N_m(i) = m_\mathrm{I} + 1 - 2N_f(i) = (p(3n_f + \delta) + q) + 1 - 2(p\, n_f + q') \tag{5.7}$$

$$= (p-1)n_f + p\,\delta + (q - q') + (n_f - q') + 1 > 1$$

Since the majority difference is greater than zero, the outputs of PE_i can be recovered correctly. Note that the proof is based on a worst-case analysis, hence the condition found is only a sufficient condition. □

Theorem 6.1 below shows the necessary and sufficient condition for recovering from incorrect outputs of a neuron in an isomorphic set.

Theorem 6.1. For an isomorphic set **I** with n_I neurons and its associated $\mathbf{HR}_\mathrm{I}$ with n_f faulty PEs, the necessary and sufficient condition for $\mathbf{HR}_\mathrm{I}$ to detect all these faulty PEs and recover the correct outputs is

$$2n_f \leq m_\mathrm{I} < n_\mathrm{I} \tag{5.8}$$

Proof. The proof follows from Lemmas 6.1 and 6.2. It is not needed to include the condition for $m_\mathrm{I} \geq n_\mathrm{I}$ because it results in a weaker sufficient condition. □

The probability of safeness of $\mathbf{HR}_\mathrm{I}$ is quantitatively described below. $\mathbf{HR}_\mathrm{I}$ is said to be *safe* if incorrect outputs generated by PEs can be recovered in the decision cells, assuming that the decision cells are operating correcting. Suppose $m_\mathrm{I} < n_\mathrm{I}$, the probability of safeness is

$$\Pr[\,\mathbf{HR_I} \text{ is safe}\,] = \Pr[\,n_f \le m_I/2\,] = F(m_I/2), \tag{5.9}$$

where F is the cumulative distribution function of the number of faulty PEs in $\mathbf{P_I}$.

Given an arbitrary distribution of the number of faulty PEs in $\mathbf{P_I}$, the degree of redundancy m_I can be determined by finding the minimal m_I such that the probability of safe operation is greater than a threshold ε. That is,

$$F(m_I/2) \ge \varepsilon \tag{5.11}$$

To illustrate the behavior of **HR**, we continue with Example 4.1 described in the previous section. Recall that $\mathbf{HR_{I_3}}$ has 14 PEs. Assume that there are four faulty PEs, *i.e.*, $n_f = 4$ (cf. $n_f = 2$ in Example 4.1). $\mathbf{HR_{I_3}}$ has different fault-tolerance behavior for different patterns of faulty PEs. Consider the following three cases: (a) all faulty PEs are adjacent to each other, (b) adjacent faulty PEs are interleaved in between by one working PE, and (c) adjacent faulty PEs are interleaved in between by two working PEs. By assuming that the correct outputs of all PEs are 1s, and that faulty PEs are stuck at 0s, Cases (a), (b), and (c) are respectively illustrated in Tables 5.1a, 5.1b, and 5.1c.

Table 5.1a. Four faulty PEs adjacent to each other.

Redundancy	Recovered Outputs by $\mathbf{HR_{I_3}}$														Recovery
Degree m_{I_3}	o_0^*	o_1^*	o_2^*	o_3^*	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	o_{11}	o_{12}	o_{13}	Index
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0.71
1	?	0	0	0	?	1	1	1	1	1	1	1	1	1	0.64
2	1	0	0	0	0	1	1	1	1	1	1	1	1	1	0.71
3	1	?	0	0	0	?	1	1	1	1	1	1	1	1	0.64
4	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0.71
5	1	1	?	0	0	0	?	1	1	1	1	1	1	1	0.64
6	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0.71
7	1	1	1	?	?	?	?	?	1	1	1	1	1	1	0.64
≥8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00

Table 5.1b. Four faulty PEs with adjacent pair interleaved by one working PE in between.

Redundancy	Recovered Outputs by $\mathbf{HR_{I_3}}$														Recovery
Degree m_{I_3}	o_0^*	o_1	o_2^*	o_3	o_4^*	o_5	o_6^*	o_7	o_8	o_9	o_{10}	o_{11}	o_{12}	o_{13}	Index
0	0	1	0	1	0	1	0	1	1	1	1	1	1	1	0.71
1	?	?	?	?	?	?	?	?	1	1	1	1	1	1	0.43
2	1	1	0	1	0	1	0	1	1	1	1	1	1	1	0.79
3	1	1	?	?	?	?	?	?	1	1	1	1	1	1	0.57
4	1	1	1	1	0	1	0	1	1	1	1	1	1	1	0.86
5	1	1	1	1	?	?	?	?	1	1	1	1	1	1	0.71
6	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0.93
7	1	1	1	1	1	1	?	?	1	1	1	1	1	1	0.86
≥8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00

Table 5.1c. Four faulty PEs with adjacent pair interleaved by two working PE in between.

Redundancy	Recovered Outputs by $\mathbf{HR_{I_3}}$														Recovery
Degree m_{I_3}	o_0^*	o_1	o_2	o_3^*	o_4	o_5	o_6^*	o_7	o_8	o_9^*	o_{10}	o_{11}	o_{12}	o_{13}	Index
0	0	1	1	0	1	1	0	1	1	0	1	1	1	1	0.71
1	?	?	1	?	?	1	?	?	1	?	?	1	1	1	0.43
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00
3	1	1	1	?	1	1	?	1	1	?	1	1	1	1	0.79
≥4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.00

An entry of "?" in Table 5.1 indicates that m_I is odd (or m_I+1 is even), that the number of correct outputs is $(m_I+1)/2$, and that the correct output cannot be recovered. The *recovery index* is the ratio of the number of correct outputs recovered by $\mathbf{HR}_{I_3}$ with m_{I_3} to the number of outputs. According to Theorem 6.1, the maximum degree of redundancy needed is $m_{I_3} = 2n_f = 8$ in this example (cf. $m_{I_3} = 6$ in Example 4.1). In cases (a) and (b), the outputs cannot be recovered until $m_{I_3} = 8$. In case (c), $m_{I_3} = 4$ is sufficient.

In general, errors due to adjacent faulty PEs are more difficult to recover and require a higher degree of redundancy. In the worst case, the degree of redundancy must be $2n_f$, as proved in Theorem 6.1. In cases in which faulty PEs are not adjacent to each other, the necessary degree of redundancy can be less than $2n_f$.

6. OVERHEAD ANALYSIS

In this section, we analyze the hardware and time needed for implementing the proposed spatial and temporal redundancies.

The cost due to spatial redundancy R_S is defined as the ratio of the amount of additional hardware needed for implementing hybrid redundancy to the amount of hardware without redundancy. The overhead due to temporal redundancy R_T is defined as the ratio of the amount of extra time needed for performing operations using hybrid redundancy to the amount of time needed in the non-redundant case.

For spatial redundancy, additional storage banks for storing weight vectors are needed. Let η_I be the ratio of the hardware for one storage bank to the hardware for a non-redundant PE in $\mathbf{HR}_I$, and s_I be the amount of hardware of a non-redundant PE in $I \in s_I$. Note that $0 < \eta_I < 1$. The cost of spatial redundancy R_S is

$$R_S = \frac{\sum_{I \in \Theta_C} m_I \, n_I \, s_I \, \eta_I}{\sum_{I \in \Theta} n_I \, s_I}, \qquad (6.1)$$

where Θ is the set of all isomorphic sets. The spatial redundancy is approximately proportional to the total number of neurons in the critical set and also to the average degree of redundancy for all isomorphic sets in the critical set. If the critical set is small as compared to the set of isomorphic sets, then the spatial redundancy is small.

The derivation of the overhead due to temporal redundancy is more complex, since multiple isomorphic sets may be operating concurrently and their effects are not additive. Assuming that the operations in the decision cells are overlapped with the operations in a PE, the overhead in time for an isomorphic set of neurons implemented using hybrid redundancy can be considered as a series of m_I layers of neurons, each implemented without redundancy. The effect on time can be analyzed by finding the critical path in both the non-redundant and redundant cases. Let p_R and p_{NR} be the lengths of the critical paths in the redundant and non-redundant cases, respectively. The overhead due to temporal redundancy is, therefore,

$$R_T = \frac{p_R}{p_{NR}} - 1 \qquad (6.2)$$

Consider Example 4.1: $\Theta = \{1, 2, 3\}$, $\Theta_C = \{3\}$, $n_{wf} = 3$, $n_{I_1}=100$, $n_{I_2}=200$, and $n_{I_3}=14$. For simplicity, let s_I be a constant s. The cost due to spatial redundancy $R_{S_{EX4.1}}$ is

$$R_{S_{EX4.1}} = \frac{\sum_{I \in \Theta_C} m_I \, n_I \, s_I \, \eta_I}{\sum_{I \in \Theta} n_I \, s_I} = \frac{6 \times 14 \times s \times \eta_{I_3}}{100 \times s + 200 \times s + 14 \times s} = 0.27 \times \eta_{I_3} \qquad (6.3)$$

The length of the critical path redundancy is proportional to 3, and the length with redundancy is proportional to 9 (m_{I_3} +3). The overhead due to temporal redundancy $R_{T_{EX4.1}}$ is

$$R_{T_{EX4.1}} = \frac{(9-3)}{3} = 2 \qquad (6.4)$$

7. CONCLUSIONS

In this paper, a hybrid redundancy model based on temporal redundancy, spatial redundancy, and coding is proposed. By recognizing that multiple neurons are receiving identical inputs in a multi-layer neural network, a fault-tolerant neural-network architecture based on hybrid redundancy is studied. Our analysis indicates that the reliability can be enhanced by increasing the degree of redundancy. We also prove the necessary and sufficient condition on the range of the degree of redundancy. The overhead and cost of implementation are relatively small as compared to other methods.

REFERENCES

[1] W. T. Cheng and J. H. Patel, "Concurrent Error Detection in Iterative Logic Arrays," *Proc. 14th Int'l Conf. on Fault-Tolerant Computing*, pp. 10-15, ACM/IEEE, 1984.

[2] DARPA, *Executive Summary of DARPA Neural Network Study*, MIT Lincoln Laboratory, Lexington, MA, July 8, 1988.

[3] P. K. Lala, *Fault Tolerant & Fault Testable Hardware Design*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[4] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *Accoustics, Speech and Signal Processing Magazine*, pp. 4-22, IEEE, April 1987.

[5] J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1*, Bradford Books (MIT Press), Cambridge, Massachusetts, 1985.

[6] W. R. Moore, "Conventional Fault-Tolerance and Neural Computers," *private communication*, Dept. of Engineering Science, Oxford Univ., Oxford, England, 1988.

[7] J. H. Patel and L. Y. Fung, "Concurrent Error Detection in ALUs by Recomputing with Shifted Operands," *Tran. on Computers*, vol. C-31, no. 7, pp. 589-595, IEEE, July 1982.

[8] D. K. Pradhan, *Fault Tolerant Computing: Theory and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.