Scientia Iranica, Vol. 9, No. 4, pp 336–345
© Sharif University of Technology, October 2002

SCIENTIA
IRANICA

A Neural Network Approach to Detect and Correct Single Faults in Nonlinear Systems

A. Moosavinia* and K. Mohammadi[1]

In this paper, a new fault tolerant feed forward neural network architecture is first introduced, using fault tolerant neural nodes in its output layer, which is the most critical layer in the network. Then, the conventional Back error Propagation (BP) algorithm is modified to apply to this architecture with the least learning degradation for fault tolerant nodes. The presented neural network, jointed with a systematic real convolutional code, contributes in an Algorithm Based Fault Tolerant (ABFT) scheme to protect a nonlinear data process block. The neural network is trained to produce an all zero syndrome sequence in the absence of any faults. The convolutional code guarantees that faults, representing errors in the processed data, will result in notable nonzero values in syndrome sequence. A majority logic decoder can easily detect and correct single faults by observing the syndrome sequence. Simulation results for random s-a-0 faults, demonstrating error detection and correction behavior, are also presented.

INTRODUCTION

Neural networks have been successfully used for fault diagnosis in nonlinear systems [1-3]. The most attractive features of neural networks described in these papers are their capability to model nonlinear systems and their ability to tolerate faults. However, recent research [4,5], shows that these networks are not really fault tolerant. Indeed, there are always many nodes in a large neural network that do not contribute to neural network function, therefore, in contrast to using redundant nodes, fault tolerance is not improved. In fact, nodes can often be found in a neural network that are too important and their failure can cause a system crash.

On the other hand, the use of conventional fault tolerant techniques such as Triple Modular Redundancy (TMR) and Triple Time Redundancy (TTR) [6], yields either a very expensive and large system or a lengthy overhead. Algorithm based fault tolerant techniques are good choices for error detection and correction in linear systems, by promoting cheap and small variations in hardware and/or software [7].

In this paper, a fault tolerant neural network architecture will first be introduced, based on Multilayer Perceptron (MLP) and a new learning algorithm, based on conventional Back error Propagation (BP) algorithm. Then, this neural network is utilized in an ABFT architecture, using convolutional codes to correct single faults in a nonlinear system [8].

Two main approaches have been proposed to improve fault tolerance in an artificial neural network: 1) Working on learning algorithms and 2) Working on architecture. Most of the reported papers deal with the learning phase or algorithm. In fact, it is believed that the distributed architecture of neural networks is not suitably utilized by current common learning algorithms, such as BP, in order to achieve or enhance fault tolerance in neural networks. In [9] this enhancement is achieved by manipulating the gradient of a sigmoid function during the learning phase. Murray and Ito [10,11] have used the well-known method of fault injection during the learning procedure and have shown that the fault behavior of neural networks can be greatly improved against stuck-at-0 and stuck-at-1 faults. Neti [12] has introduced a network called "maximally fault tolerant neural network", in which its weight coefficients are estimated through a nonlinear optimization problem to get the maximum allowable fault tolerance in neural network. There are few reports considering neural network architecture for improving fault tolerance. Zhang [13] has studied feedback neural networks with hard limiting output. The results show that the fault tolerance of such networks cannot be

*. *Corresponding Author, College of Electrical Engineering, Iran University of Science and Technology, Tehran, I.R. Iran.*

1. *College of Electrical Engineering, Iran University of Science and Technology, Tehran, I.R. Iran.*

improved through adding more nodes. Chiu [14] addressed some modifications of the architecture, such as addition/deletion nodes, however, the paper is still based on the learning procedure. Kwon [15] presented a method to break critical nodes in a trained MLP, to have a predefined level of fault tolerance. Considering the assumption that the weights of all links are known and stored in a memory, Tanprasert [4] modified a two-layer feed forward neural network to detect faulty links.

In the next sections of this paper, first, the ABFT concept is introduced briefly. Then, the convolution code used in this paper is described and, a multilayer perceptron network with conventional BP algorithm, is presented. Moreover, fault model and sources in a neural network and the modified architecture and learning algorithm are introduced, respectively. Simulation details and results are provided and, finally, the main advantages of the proposed method are concluded.

ABFT SCHEME

ABFT has been suggested to design fault tolerant array processors and systolic array systems. The scheme is capable of detecting and, sometimes, correcting errors caused by permanent or transient faults in the system. It was first proposed as a checksum approach for matrix operations [16,17]. Since then, the technique has been extended to many digital signal processing applications, such as Fast Fourier Transform [18,19], linear and partial differential equation solving [20,21], digital filters [22] and the protection of linear [23] and general multiprocessor systems [24].

The basic architecture of an ABFT system is shown in Figure 1. Existing techniques use various coding schemes to provide information redundancy, needed for error detection and correction. As a result, this encoding/decoding must be considered as the overhead introduced by ABFT.

The coding algorithm is closely related to the running process and is often defined by real number codes generally of the block type [25]. Systematic codes are of most interest because the fault detection scheme can be superimposed on the original process box, with fewest changes in algorithm and architecture.

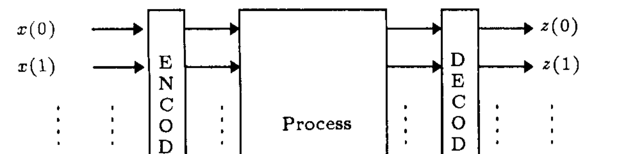

Figure 1. ABFT general architecture.

In most previous ABFT applications, the process to be protected is often a linear system. In this paper, a more common case is assumed consisting of linear or nonlinear systems but still constrained to static systems. This assumption is due to selecting a static neural network in the main architecture.

Convolutional Codes

A convolutional encoder processes the data stream sequentially and, for every k information symbol presented to it, there are $n(n > k)$ output symbols. Hence, $n - k$ parity codes are generated. The coding scheme depends on the history of a certain number of input symbols. The total register length used in the decoder is called the constraint length. This code has been used as a suitable mechanism in data communication for many years [26] and, although they are basically designed to protect data streams on finite fields, research on infinite fields is also reported [27]. Only systematic forms of convolutional codes are considered because the normal operation of the Process block is not altered and there is no need for decoding in order to obtain true output. In addition, systematic convolutional codes are proved to be noncatastrophic.

The generator matrix of a systematic convolutional code, G, is a semifinite matrix involving m finite submatrixes as:

$$G=\begin{bmatrix} IP_0 & 0P_1 & 0P_2 & \cdots & 0P_m & & \\ & IP_0 & 0P_1 & \cdots & 0P_{m-1} & 0P_m & \\ & & IP_0 & \cdots & 0P_{m-2} & 0P_{m-1} & \cdots \\ & & & & \cdots & & \\ & & & & & \cdots & \cdots \end{bmatrix}, \tag{1}$$

where I and 0 are identity and all zero $k \times k$ matrixes, respectively [28] and P_j with $j = 0$ to m is a $k \times (n-k)$ matrix, whose entries are:

$$P_j = \begin{bmatrix} g_{1,j}^{k+1} & g_{1,j}^{k+2} & \cdots & g_{1,j}^{n} \\ g_{2,j}^{k+1} & g_{2,j}^{k+2} & \cdots & g_{2,j}^{n} \\ \cdots & \cdots & \cdots & \cdots \\ g_{k,j}^{k+1} & g_{2,j}^{k+2} & \cdots & g_{k,j}^{n} \end{bmatrix}. \tag{2}$$

Unfilled areas in G matrix indicate zero values. The parity check matrix associated with this code is given by:

$$H = \begin{bmatrix} P_0^T I & & & & \\ P_1^T 0 & P_0^T I & & & \\ \cdots & \cdots & \cdots & & \\ P_m^T 0 & P_{m-1}^T 0 & \cdots & P_0^T I & \\ & P_m^T 0 & \cdots & P_1^T 0 & \cdots \\ & & & \cdots & \cdots \end{bmatrix}, \tag{3}$$

where, in this case, I and 0 are identity and all zero $(n - k) \times (n - k)$ matrixes, respectively and T is the transpose sign. The syndrome equations, denoted by vector S, are given by:

$$S = rH^T = eH^T, \tag{4}$$

where r is the received sequence and e is error pattern. When r is a code word, S is zero, else it has some non-zero values.

There are three principal ways of decoding convolutional codes, viterbi, sequential and majority-logic decoding [25]. The viterbi algorithm is an optimal decoding procedure based on the maximum likelihood approach but it requires 2^k computations per decoded information bits. On the other hand, it has a decoding delay equal to the information frame length, so it consumes a large amount of memory and computation time. Sequential decoding is a near optimal scheme with an average of 1 or 2 computations per information bit, but it still has a delay as long as the input data stream. A majority-logic decoder, on the other hand, has the least performance but needs only one constraint length of code and one computation per bit for decoding, thereby, minimizing memory usage and enhancing decoding speed. This paper, therefore, uses majority-logic decoding for its convolutional code.

Self Orthogonal Codes

Majority logic decoding is based on orthogonal parity-check sums, that is, the equation relates any syndrome bit or any sum of those bits to channel error bits. By definition, a set of J such summations are orthogonal on an error bit, e_j, if each sum contains e_j but no other error bit is in more than one check sum equation. The majority-logic decoding rule says that the estimated error bit, $\hat{e}_j$, is 1 if more than $t_{ML} = \lfloor J/2 \rfloor$ of J orthogonal check sums have a value of 1. t_{ML} is called the majority-logic correcting capability of the code [25].

To employ the maximum error correcting capability of the code, it must be completely orthogonalizable [25], that is, a code in which $J = d_{\min} - 1$. By definition $d_{\min}$ is:

$$d_{\min} = \min\{w[v]_m : u_0 \neq 0\}, \tag{5}$$

where v is a code word and u_0 is the first nonzero input information sequence. Note that $d_{\min}$ is calculated over the first constraint length of the code.

Self-orthogonal codes are one class of codes that are completely orthogonalizable. In such a code, for each information error bit, the set of all syndrome bits that involve that bit form an orthogonal check set on that bit, without adding syndrome bits. Using these codes makes easier implementation of majority-logic decoding.

MLP AND BP ALGORITHM

MLP consists of several cascaded layers of neurons with sigmoid activation functions [29]. The input vectors feed into each of the first layer neurons, then the output of this layer feeds into each of the second layer neurons and so on, as shown in Figure 2.

Most often, the nodes are fully connected, that is, every node in layer l is connected to every node in layer $l+1$. In this paper, the input vector is assumed as the first layer in the neural network. MLP can easily perform Boolean logic operations, pattern recognition, classification and nonlinear function approximation [30]. Usually, output neurons use linear activation functions rather than nonlinear sigmoid, since this tends to make learning easier. MLP is a supervised neural network that learns through examples and the most common learning algorithm used for it is BP, which is the steepest descend gradient based algorithm. In this paper, it is assumed that the number of hidden layers is one and the activation function of each neuron in the hidden layer is a bipolar sigmoid signified by the following equation:

$$f(u_i) = \frac{1 - \exp(-u_i)}{1 + \exp(-u_i)}, \tag{6}$$

$$u_i = \sum_j w_{ij} \times x_i - \theta_i, \tag{7}$$

where w_{ij} is the connection weight between neuron j in the preceding layer and neuron i. x_j is neuron j's output and θ_i is a bias value for the neuron i. Back-propagation algorithm changes w_{ij}, in order to reduce the error of output layer defined by:

$$E = \frac{1}{2}\sum_i (t_i - o_i)^2, \tag{8}$$

where t_i is ith output target and o_i is the ith estimated output [31].

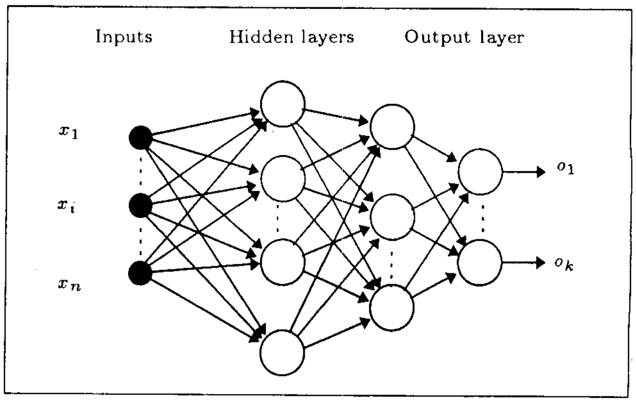

Figure 2. Architecture of a typical MLP.

Using the steepest-descent gradient rule, the change of w_{ij} is expressed as:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \tag{9}$$

η is a positive real number called "learning rate" that determines step size in w_{ij} changes. Selecting a suitable η value plays an important role in network learning convergence [32].

Back-propagation algorithm says that:

$$\Delta w_{ij} = \eta \delta_i^P o_i^P, \tag{10}$$

$$\delta_i^P = (t_i^P - o_i^P).f'(u_i^P), \tag{11}$$

$$\delta_i^P = (\sum_k w_{ki}.\delta_k^P).f'(u_i), \tag{12}$$

where Equation 11 is for an output layer and Equation 12 is for neurons in the hidden layer. $f'()$ is the derivation of the bipolar sigmoid and is calculated by:

$$f'(x) = 2f(x)(1 - f(x)). \tag{13}$$

FAULT MODEL

There are usually three kinds of fault considered in a neural network: 1) Connection, 2) Weights and 3) The neuron body itself [11]. The first two faults are often modeled as s-a-0 (stuck at 0) and most often occur during a memory disappearance or a link disconnection in VLSI. On the other hand, faults due to a neuron cell, usually subject its output to one of the positive or negative saturation voltages. This kind of fault is modeled as s-a-(1) (stuck at one) or s-a-(-1). In this paper, only s-a-0 faults are considered.

FTNN ARCHITECTURE

Experiments show that output neurons are the critical nodes in an MLP network. Table 1 summarizes the effect of s-a-0 faults in the links of an MLP trained by a standard BP algorithm. The network consists of 4 inputs, 7 neurons in a hidden layer and one output node and is trained to approximate a nonlinear function by the following equation:

$$\text{Out} = 0.25 \times (x + y + z + w)^2. \tag{14}$$

Clearly, links in the first layer have tolerated faults better than those in the output layer, which are sensitive and can cause remarkable errors. So, replacing the conventional output nodes with non-faulty nodes will improve the total fault behavior in an MLP network.

Table 1. The effect of 10000 random s-a-0 faults in first and second layer of a 4-7-1 MLP trained by BP algorithm.

Fault Location	No Faults	First Layer	Second layer
Average utput error	0.0001	0.0224	0.3359

To obtain such a robust node in the output layer, the use of a linear activation function is first suggested, which is a simple adder indeed, rather than the nonlinear sigmoid one. Then, a wired connection is presented as a link to eliminate memory usage by the elimination of connection weights. Figures 3a to 3d suggest how to obtain such nodes in two steps, while preserving, as much as possible, the learnability of the neural network in output nodes. The resultant node is called FTN (Fault Tolerant Neuron). Utilizing FTN nodes in the output layer of an MLP yields an architecture which is called FTNN (Fault Tolerant Neural Network). FTNN is shown in Figure 3d, in which hidden layer nodes are conventional nonlinear nodes but the output consists of FTN nodes.

Figure 3a shows a conventional node in an MLP with a nonlinear activation function, f, and connection weights of w_1 to w_n. Usually, a BP algorithm yields to non equal w_i values. According to FTN architecture, an algorithm will be introduced to produce weights with close magnitudes, but which may have different signs. This is done by moving the weights, gradually, to a unique value and adapting the learning rate, η. In the next section, the whole algorithm will be described in detail. Figure 3b shows such a trained node. W is the magnitude value obtained through the algorithm.

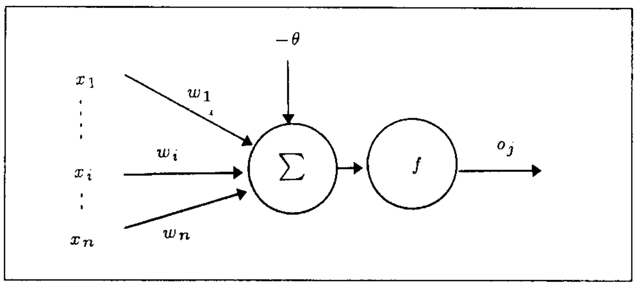

Figure 3a. A conventional nonlinear node .

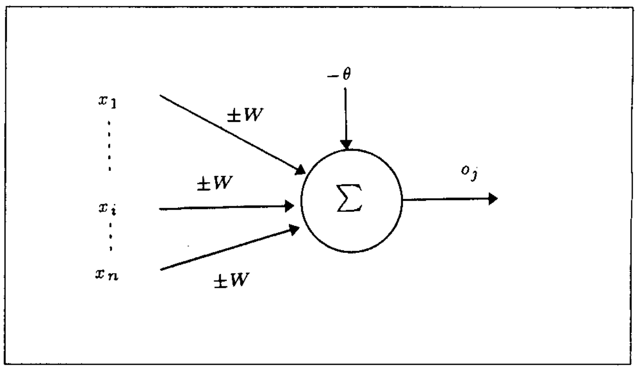

Figure 3b. A linear node with similar input weights.

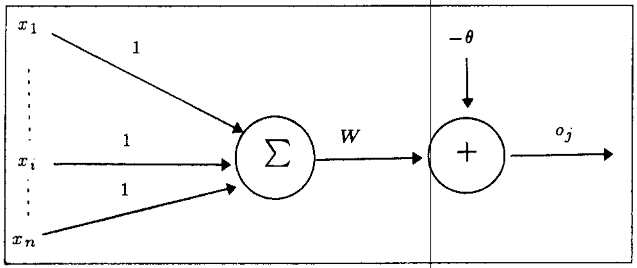

Figure 3c. Fault tolerant node with wired input connections and a single weight.

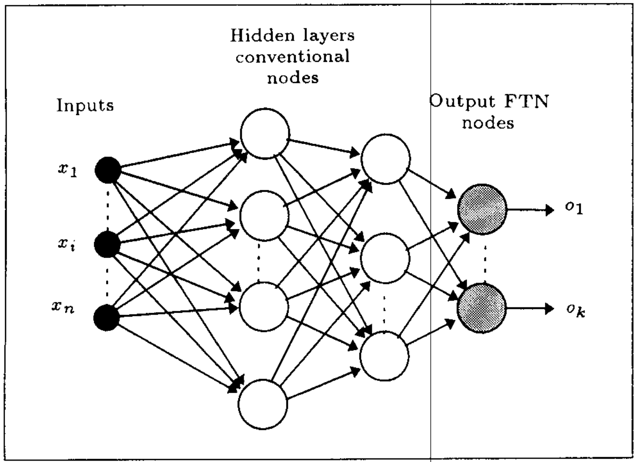

Figure 3d. An FTNN architecture using FTN nodes in output layer (shadowed nodes).

Then, the connection coefficients are shifted to the adder part and factored by the magnitude term, W, so that the remaining node has n wired connections as links, an adder that performs subtraction as well and a single weight of W.

UWLA LEARNING ALGORITHM

According to the architecture described in the previous section, a learning algorithm is introduced for the output layer, which is called UWLA (Uniform Weight Learning Algorithm). For an output node, the algorithm results in a unique weight coefficient called W, and the goal is to have uniform weight distribution in the MLP architecture. In fact, the standard BP algorithm described in the previous section does not guarantee a uniform usage of architecture, because its goal is to have a minimum sum of square errors, E, defined by Equation 8. Here, another criterion is added to the learning process, i.e., the variance of w_{ij}s, the weights of output node j. The aim is to make this term as close as possible to zero. The variance is defined as:

$$\sigma_j = \sum_i (w_{ij} - m_j)^2, \tag{15}$$

where j and i denote the neurons in the output and hidden layer, respectively and m_j is the mean value of w_{ij} for output neuron j and is calculated after each training step. Equation 10 shows that Δw_{ij} is related to η, the learning rate parameter. η, will be modified to adopt a smaller value when Δw_{ij} increases variance, and a larger value when it is in the direction of variance decrease. The following algorithm is suggested:

UWLA Algorithm:

```
Step 0: initialize weights with small values;
Step 1: while stopping condition is false, do steps 2-9;
Step 2: for each input vector, do Steps 3-8;
Step 3: each input unit receives input signal and
        broadcasts it to hidden layer units;
Step 4: each hidden unit sums its weighted input signal
        and applies its activation function according to
        Equations 7 and 8;
Step 5: each output unit sums its weighted input signal
        and produces its output, too.
Step 6:
Step 6a: for each output unit, compute its error infor-
         mation term, using Equation 11;
Step 6b: calculate mean $[w_{ij}(\text{old})]$;
Step 6c: choose $\eta = \eta 1$. Calculate:
         $w_{ij}(new)=w_{ij}(old)+\Delta w_{ij}\eta$ using Equation 10;
         if $abs[w_{ij}(\text{ new}) - \mathbf{mean}[w_{ij}(\text{old})]]$ is greater
         than $abs[w_{ij}(\text{old}) - \mathbf{mean}[w_{ij}(\text{old})]]$;
          choose $\eta = \eta 2$, where $\eta 2 < \eta l$.
Step 7: for each hidden unit, compute its error infor-
        mation term, using Equation 12;
Step 8: for each output and hidden unit, update
        weights, according to:
          $w_{ij}(\text{new}) = w_{ij}(\text{old}) + \Delta w_{ij}\eta$.
Step 9: test stopping condition.
```

FTNN Behavior

To evaluate the fault tolerance of an MLP, a measure of sensitivity is first presented as:

Definition 1

$S(w_i)$, the sensitivity of a neural network to weight w_i, is the effect on Mean Square Error (MSE) when w_i is forced to zero.

As an example, an MLP network with four inputs, seven hidden nodes and one output node is trained with UWLA to approximate the nonlinear function of Equation 14.

Training samples are non-correlated random numbers from $T_L = \{0, 0.1, 0.2, ..., 0.9, 1.0\}$. For the comparison proposed, the standard BP and the MFTA (Multiple Fault Training Algorithm) algorithm introduced by [13] are also considered. MFTA is based on the well-known method of fault injection during the learning process. In this example, a s-a-0 fault is injected in the hidden layer nodes for every 500 iteration. For all simulations, the training process lasts after 100000 iterations with a learning rate of $\eta = 0.005$. UWLA uses the second learning rate $\eta_2 = 0.0025$.

The resultant weights for UWLA are listed in Table 2. The single weight $W = 0.786674$ has been chosen as the average of all the weights in the second layer. All trained networks are then subjected to s-a-0 faults in their second layer. In the testing phase, inputs are selected from the set $T_T = \{0, 0.01, 0.02, ..., 0.99, 1.00\}$. Table 3 shows that the network trained with the UWLA has the largest average error, that is 0.0074, for the testing set, when there is no fault injected in its second layer. BP and MFTA have a better response in this case and their average error is less than 0.0001. However, as s-a-0 faults in the hidden node outputs are introduced, UWLA will show its benefits. The calculated sensitivity measure for all nodes in the hidden layer are also shown in Table 3. m_f is the average sensitivity for each network. Clearly, the network trained with UWLA has the least sensitivity with a value of 0.0831. The sensitivity of the network trained with MFTA is 0.2332, while the network trained with conventional BP has the largest sensitivity, as high as 0.3354.

In hardware implementation, the FTNN architecture will show its extra benefits for using fault tolerant nodes in its output layer, although the described simulation does not consider hardware features.

FAULT CORRECTION USING FTNN

Convolutional codes are usually used over the transmission channels, through which both information and parity bits are sent. To achieve fault detection and obtain the correction properties of this code in a nonlinear process, with minimum overhead computations, the block diagram in Figure 4 is proposed.

The main architecture is similar to a normal ABFT scheme, except for the FTNN and the delays in the information pass, which replace the parity generator part of a systematic convolutional encoder. The upper way is the normal process data flow, which passes through the nonlinear process block and is then fed to the convolutional encoder to make parity sequence y''. On the other hand, FTNN is trained to have a direct parity, y', equal to y'', in the absence of any noise and faults in the system, using UWLA. So, the syndrome sequence is a stream of zero, or near zero, values in a normal operation.

The faults in a nonlinear process block have been modeled with module noise A, while the encoder and neural network noises are modeled with modules B and C. Since these last two noises contribute to the

Table 2. Weights of a 4-7-1 MLP trained by WLA.

Hidden	**Hidden Node 1**	**Hidden Node 2**	**Hidden Node 3**	**Hidden Node 4**	**Hidden Node 5**	**Hidden Node 6**	**Hidden Node 7**
Input node 1	-0.2845	0.1730	0.6452	0.3863	-0.2459	0.3882	0.0832
Input node 2	-.02498	0.3436	0.3511	0.2596	0.0087	0.2352	0.3051
Input node 3	0.4929	0.2290	-0.2649	0.2361	0.6337	-0.1338	-0.0616
Input node 4	0.2406	0.0346	-0.3288	0.4759	0.2460	0.6359	-0.1767
Hidden node bias	0.3455	-0.5955	0.1151	-1.1039	0.4664	-0.8699	0.8099
Output node	0.7863	0.7869	0.7864	0.7872	0.7868	0.7870	0.7860
Output node bias	0.6858						

Table 3. Sensitivity measures of networks trained with different algorithms to s-a-0 faults in hidden nodes.

		Hidden Node No.							**Average**
Algorithm	**No Fault**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	m_f
BP	0.0001	0.5835	1.0617	0.0960	0.0839	0.0457	0.0576	0.4195	0.3354
MFTA	0.0001	0.5911	0.6128	0.0711	0.0473	0.0317	0.0397	0.2384	0.2332
UWLA	0.0072	0.0697	0.0306	0.0595	0.0852	0.0311	0.0604	0.2449	0.0831

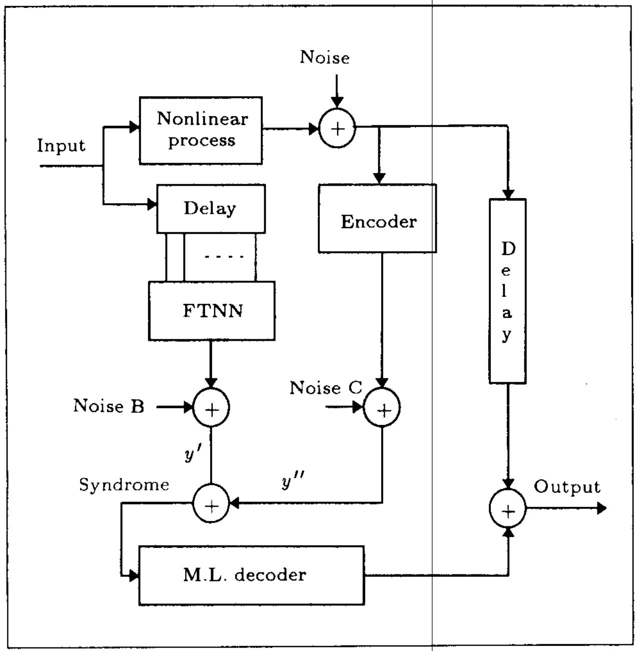

Figure 4. Block diagram of the neural based ABFT.

syndrome additively, one of them may be deleted without any degradation.

Example and Simulations

A $(3,2,2)$ systematic convolutional code with generators of:

$$g_1^{(3)} = 1 + D, \tag{16}$$

$$g_2^{(3)} = 1 + D^2, \tag{17}$$

is used to evaluate the error detectability and correctability of the proposed method. An FTNN with 4 inputs, 10 nodes in the hidden layer and one output is selected. The single output node of this NN is an FTN trained by UWLA and all other nodes are conventional nodes with a nonlinear sigmoid function. A block processing SINE function with two inputs is chosen as the nonlinear process to be protected. The generator matrix of the code, for its first constraint length, is as:

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ & & & 1 & 0 & 1 & 0 & 0 & 1 \\ & & & & 1 & 1 & 0 & 0 & 0 \\ & & & & & & 1 & 0 & 1 \\ & & & & & & & 1 & 1 \end{bmatrix}, \tag{18}$$

and, according to Equation 3, the parity check matrix for the first constraint length is:

$$H = \begin{bmatrix} 1 & 1 & 1 & & & & & & \\ 1 & 0 & 0 & 1 & 1 & 1 & & & \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \tag{19}$$

There are two parity triangles for each generator, as:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ 1 & 1 & \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} e_0^{(1)} \\ e_1^{(1)} \\ e_2^{(1)} \end{bmatrix} + \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_0^{(2)} \\ e_1^{(2)} \\ e_2^{(2)} \end{bmatrix}. \tag{20}$$

Clearly, the code is self orthogonal and a set of two orthogonal check sums can be formed on the information error bit, hence, $t_{ML} = 1$ and the code can correct single faults in each constraint length of code which, here, is three. Figure 5a shows the main process, that is a two-input SINE block. The outputs y_1 and y_2, are subjected to uncorrelated single s-a-0 faults, modeled with noise modules A1 and A2. The faulty outputs now shown with b_1 and b_2, are then fed to the convolutional encoder as in Figure 5b. The code stream generated here, y'', is compared with FTNN output y' to produce the syndrome sequence as shown in Figure 5c. Considering the syndrome sequence, the majority logic produces two error signals that are fed back to delayed output streams in Figure 5a to correct outputs.

$\mathbb{Y}_1$ and $\mathbb{Y}_2$ are corrected outputs, in which their validity is governed by a majority logic decoding rule.

The FTNN training samples are selected randomly from [-1 1]. The network is then trained with UWLA in the absence of any noise, so that its output approximates the SINE block function followed by the (3,2,2) convolution encoder. Resulted weights for the hidden and output layer are shown in Tables 4 and 5, respectively. Weights in output layer are almost equal,

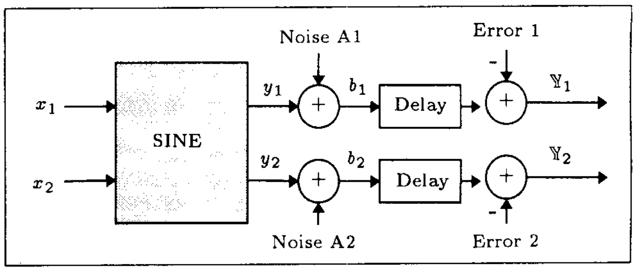

Figure 5a. SINE block input-output.

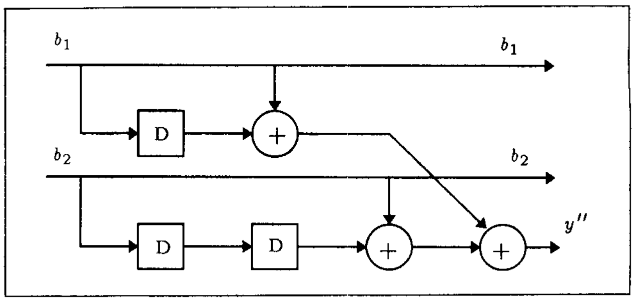

Figure 5b. Systematic (3,2,2) convolutional encoder (D denotes delay).

Table 4. Hidden layer weights and bias for 4-10-1 MLP trained by UWLA.

	Hidden Node 1	Hidden Node 2	Hidden Node 3	Hidden Node 4	Hidden Node 5	Hidden Node 6	Hidden Node 7	Hidden Node 8	Hidden Node 9	Hidden Node 10
Input node 1	0.0676	0.2662	1.6716	0.0628	-0.0888	0.3062	0.4812	-0.1105	0.6760	0.2276
Input node 2	1.0455	0.2788	0.0263	0.3283	0.5236	0.3935	0.0328	-0.0520	0.3364	0.7317
Input node 3	-0.1638	0.3458	0.0139	0.8808	0.6307	0.4505	0.1511	0.4301	0.3161	0.6321
Input node 4	0.5372	0.4723	0.1778	0.6328	0.0921	0.1249	0.4114	0.9277	-0.0108	0.2473
Bias	-0.1987	0.0709	0.0842	0.4660	-1.0564	-0.3935	-0.7360	-0.5897	-0.2929	0.4197

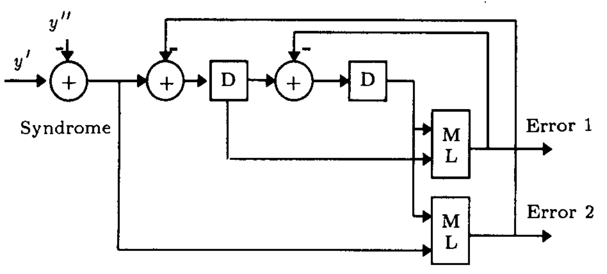

Figure 5c. Majority logic decoder used in this paper. ML is the majority logic gate.

Table 5. Output layer weights and bias for 4-10-1 MLP trained by UWLA.

	Output Node
Weights	0.5656
	0.5638
	0.5630
	0.5639
	0.5637
	0.5641
	0.5634
	0.5624
	0.5639
	0.5648
Bias	0.5649

however, their average value, 0.5639, is chosen after a training process.

Figure 6 shows the simulation results after 50000 UWLA training iterations. In Figure 6a, the syndrome sequence in the absence of any faults is shown and nonzero values are due to neural network limited accuracy. The syndrome in most cases is less than 0.005. However, there are also a few errors as great as 0.01. Therefore, a threshold value of 0.05 is selected for error detection purposes.

The whole system is then subjected to s-a-0 faults every 10 steps at SINE block outputs (module noise A1 or A2 in Figure 5a) and FTNN or encoder output (module noise C or B in Figure 4). Figure 6b shows the syndrome sequence for faults in the first output of SINE block, y_1. Clearly, there are two equal nonzero values after each fault occurrence. In Figure 6c, the same sequence is shown for faults in the second output of SINE block, y_2. Again, there are two nonzero values for each fault but the recent values are separated by a single space gap. Figure 6d shows, on the other hand, the syndrome for faults in the encoder and neural net blocks. It is clear that the syndrome contains just single nonzero values for injected faults. Therefore, there are three distinct patterns for each fault source, that is the benefit of using convolutional

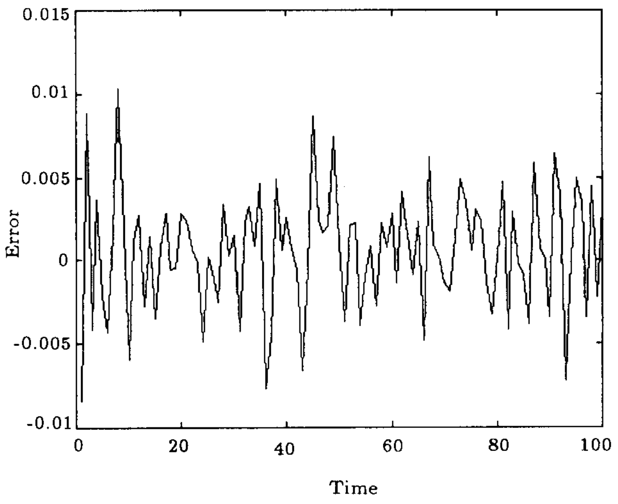

Figure 6a. Syndrome sequence for no fault condition.

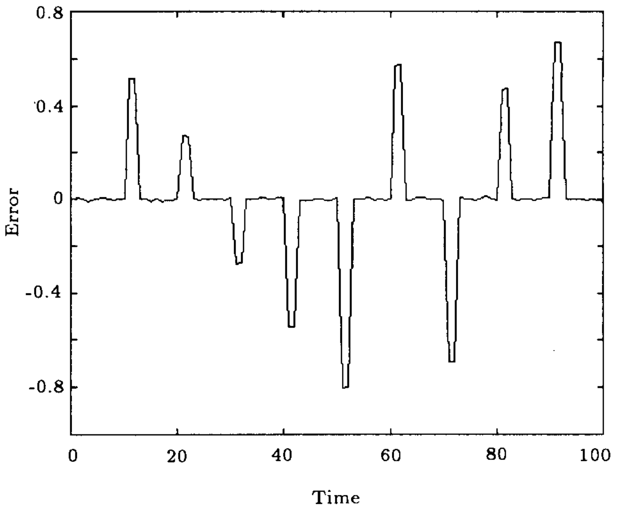

Figure 6b. Syndrome sequence introduced by noise source A1.

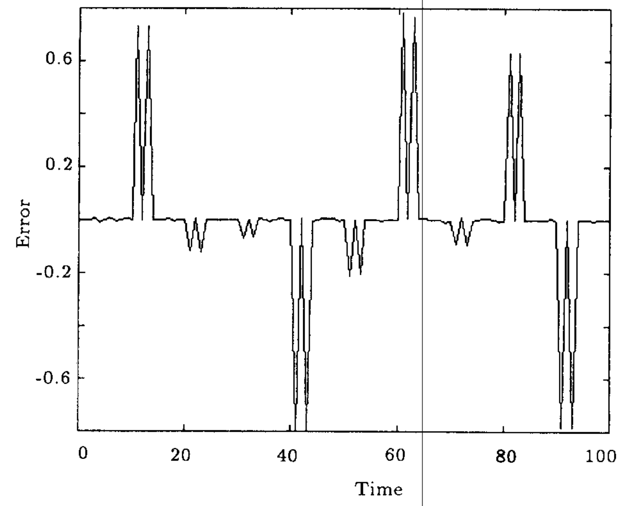

Figure 6c. Syndrome sequence introduced by noise source A2.

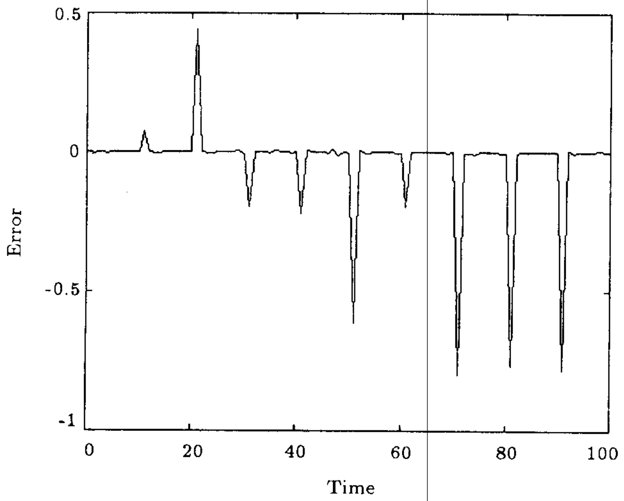

Figure 6d. Syndrome sequence introduced by noise source C or B.

Table 6. Output average error in SINE block with and without coding after 1000 injection random s-a-0 faults.

Condition	Average Error
Without coding	0.4750
With coding	0.0369

codes. If there is only a single fault in every constraint length, the majority logic decoding can correct the fault. Table 6 shows that using FTNN and ABFT architecture has reduced the average output error from 0.479 to 0.0369.

CONCLUSIONS

In this paper, an FTNN (Fault Tolerant Neural Network) architecture was first developed by implementing fault tolerant nodes, only in the output layer, which are the critical nodes in MLP networks. Then, UWLA was introduced, which is a learning algorithm that can extend back-error propagation to this new architecture. It is demonstrated that the resultant network shows a superior performance over the standard BP and commonly used fault injection training algorithms such as MFTA. Coupling FTNN and a convolutional encoder in an ABFT scheme demonstrate the feasibility and expandability obtained for fault detection and correction in nonlinear block processes. As the FTNN can itself tolerate single faults and because the majority logic gates used are very simple, unlike the other ABFT techniques, there is no need to apply extra hardware or software to protect the modules added to the main process. In addition, neural network learnability permits changing the process block functionality without severe considerations, which cannot easily be obtained through conventional ABFT techniques.

REFERENCES

1. Vukic, Z., Pavlekovic, D. and Ozbolt, H. "Rudder servo-system fault diagnosis using neural network fault modeling", *Ocean'98 Conference Proceedings*, **1**, pp 538-543, IEEE (1998).
2. Berneiri, A., D'Apuzzo, M. and Sansone, L. "A neural network approach for identification and fault diagnosis on dynamic systems", *Instrumentation and Measurment Technology Conference*, pp 564-569 (May 1993).
3. Tyan, C.U., Wang, P. and Bahler, D.R. "Neural fault diagnosis and fuzzy fault control for a complex linear dynamic system", *Int. Joint Conf. on the Fourth IEEE International Conf. on Fuzzy Systems* (1995).
4. Tanprasert, T. and Tanprasert, C. "Probing technique for neural net fault detection", *International Conference on Neural Networks*, ICNN'96 (1996).
5. Khunasaraphan, C., Vanapipat, K. and Lursinsap, C. "Weight shifting techniques for self-recovery neural networks", *IEEE Trans. on Neural Networks*, **5**(4) (July 1994).
6. Johnson, B.W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison Wesley Pub (1989).
7. Yajnik, S. and Jha, N. "Analysis and randomized design of algorithm based fault tolerant multiprocessor system under an extended model", *IEEE Tran. on Parallel and Distributed Systems*, **8**(7) (July 1997).
8. Moosavinia, A. and Mohammadi, K. "An ABFT for FIR systems using a time delay neural network", *Proceeding of 4th International Wireless and Telecommunications Symposium*, IWTS2000, pp 244-248 (May 2000).
9. Taniguchi, Y. and Kamiura, N. "Activation function manipulation for fault tolerant feed forward neural networks" , *Proceeding of the Eight Asian Test Symposium*, IEEE (1998).
10. Murray, A.F. and Edwards, P.J. "Synaptic weight noise during multilayer perceptron training: Fault tolerance and training improvement ", *IEEE Trans. on Neural Networks*, **4**(4) (July 1993).

11. Ito, T. and Takanami, I. "On fault injection approaches for fault tolerance of feed forward neural networks", *Proceedings of Sixth Asian Test Symposium*, pp 88-93 (1997).

12. Neti, C., Schneider, M.H. and Young, E.D. "Maximally fault tolerant neural networks", *IEEE Transaction on Neural Networks*, **3**(1) (Jan. 1992).

13. Zhang, T., Hu, D. and Yang, S. "Fault tolerant analysis of feedback neural networks with threshold neurons", *Proceeding of the Eighth Asian Test Symposium*, IEEE (1998).

14. Chiu, C.T., Mehrotra, K., Mohan, C.K. and Ranka, S. "Training techniques to obtain fault tolerant neural networks", *24th International Symposium on Fault Tolerant Computing* (1994).

15. Kwon, O.J. and Bang, S.Y. "Design of fault tolerant multilayer perceptron with desired level of robustness", *Electronic Letters*, **33**(12) (1997).

16. Anfinson, C.J. "A linear algebraic model of algorithm based fault tolerance", *IEEE Transaction on Computers*, **37**(12), pp 1599-1604 (Dec 1988).

17. Nair, V.S.S. and Abraham, J.A. "Real-number codes for fault tolerant matrix operations on processor arrays", *IEEE Trans. on Computers*, **39**(4) (April 1990).

18. Tao, D.L. and Hartmann, C.R. "A novel concurrent error detection scheme for FFT networks", *IEEE Transaction on Parallel and Distributed Systems*, **4**(2) (Feb. 1993).

19. Sying, Tyan Wang "Algorithm based fault tolerant for FFT networks", *IEEE Transaction on Computing*, **43**(7) (July 1994).

20. Balasubramanya, K.N., Bhuvaneswari, K. and Siva Ram Murthy, C. "A new algorithm based on given rotations for solving linear equations on fault tolerant mesh connected processors", *IEEE Trans. on Parallel and Distributed Systems*, **9**(8) (Aug. 1998).

21. Chowdhury, A.R. and Banerjee, P. "Algorithm based fault location and recovery for matrix computations on multiprocessor systems", *IEEE Trans. on Computers*, **45**(11) (Nov. 1996).

22. Robert Redinbo, G. and Zagar, B. "Modifying real convolutional codes for protecting digital filtering systems", *IEEE Trans. on Information Theory*, **39**(2) (Mar. 1993).

23. Jan-Lung Sung and Redinbo, G.R. "Algorithm based fault tolerant synthesis for linear operations", *IEEE Transaction on Computers*, **45**(4) (April 1996).

24. Blough, D.M. and Pelc, A. "Almost certain fault diagnosis through algorithm based fault tolerance", *IEEE Trans. on Parallel and Distributed Systems*, **5**(5) (May 1994).

25. Baylis, J., *Error-Correcting Codes: A Mathematical Introduction*, Chapman and Hall LTD (1998).

26. Sklar, B. "A tutorial on convolutional coding for M-Ary signals, trellis coded modulation", *Military Communications Conference*, pp 637-645 (1988).

27. Redinbo, G.R. and Zagar, B. "Modifying real convolutional codes for protecting digital filtering systems", *IEEE Trans. on Information Theory*, **39**(2) (March 1993).

28. Viterbi, A.J. and Omura, J.K., *Principles of Digital Communication and Coding*, Mc-Grawhill , 2nd Print (1985).

29. Fausell, L., *Fundamental of Neural Networks*, Printice Hall (1994).

30. Hush, D.R. and Horne, B.G. "Progress in supervised neural networks", *IEEE Signal Processing Magazine* (Jan. 1993).

31. Beale, R. and Jackson, T., *Neural Computing: An Introduction*, Department of Computer Science, University of YORK, IOP Publishing LTD (1990).

32. Adibi, A., Moosavinia, M. and Moallem, P. "Fast neural network learning using a variable learning parameter value", *3rd Iranian Con. on Electrical Engineering* (1995).