A generalized ABFT technique using a fault tolerant neural network

A. Moosavienia and K. Mohammadi

Abstract: In this paper we first show that standard BP algorithm cannot yeild to a uniform information distribution over the neural network architecture. A measure of sensitivity is defined to evaluate fault tolerance of neural network and then we show that the sensitivity of a link is closely related to the amount of information passes through it. Based on this assumption, we prove that the distribution of output error caused by s-a-0 (stuck at 0) faults in a MLP network has a Gaussian distribution function. UDBP (Uniformly Distributed Back Propagation) algorithm is then introduced to minimize mean and variance of the output error. Simulation results show that UDBP has the least sensitivity and the highest fault tolerance among other algorithms such as WRTA, N-FTBP and ADP. Then a MLP neural network trained with UDBP, 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. A systematic real convolution 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 demonstrating the error detection and correction behavior against random s-a-0 faults are presented too.

Keywords: fault tolerance, back propagation, MLP network, function approximation, ABFT, convolutional codes, majority logic decoding.

1 Introduction

One of the most attractive features of neural networks is their capability to model nonlinear systems in addition to their intrinsic fault tolerant ability. In fact neural networks have been successfully used for fault diagnosis in nonlinear systems [1], [2], [3]. However recent researches [4], [5], [31], show that these networks are not really fault tolerant. Indeed, there are always many nodes in a large neural network that do not contribute in neural network function, so in contrast to using redundant nodes, fault tolerance is not improved. On the other side, we can often find nodes that are too important and their failure can cause a system crash.

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

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

Two main approaches have been proposed to improve fault tolerance in an artificial neural network: 1) modified learning algorithms and 2) modified architectures. Most of the reported papers deal with learning phase or algorithm. In fact, it is believed that distributed architecture of neural networks is not suitably utilized by current common learning algorithms such as BP, in order to have or enhance fault tolerance in neural networks. In [8] this enhancement is achieved by manipulating the gradient of sigmoid function during learning phase. [9] and [10] have used the well known method of fault injection during learning procedure and shown that fault behavior of neural network can be greatly improved against stuck-at-0 and stuck-at-1 faults. [11] have introduced a network called "Maximally Fault Tolerant neural Network", which its weight coefficients are estimated through a nonlinear optimization problem to get the maximum allowable fault tolerance in the neural network. There are few reports considering the neural network architecture to improve fault tolerance. [12] studied feedback neural networks with hard limiting outputs. The results show that fault tolerant of such networks can not be improved through adding more nodes. [13] addressed some modification on architecture such as addition/deletion nodes but it is still based on learning procedure. In [14] a method is presented to break critical nodes in a trained MLP to have a predefined level of fault tolerance. In [4] a two layer feed forward neural network is modified to detect faulty links based on the assumption that the weights of all links are known and stored in a memory.

The following section of this paper introduces briefly the ABFT concepts. Section 3 describes the convolution code used in this paper. Then in section 4 a Multilayer Perceptron network with conventional BP algorithm in presented. In section 5 we introduce fault model and sources in a neural network. Section 6 and 7 contain our modified architecture and learning algorithm respectively. Simulation details and results are provided in section 8 and finally section 9 concludes the main advantages of the proposed method.

Iranian Journal of Electrical & Electronic Engineering, 2005.
Paper first received 11th April 2002 and in revised from 15th September 2004.
A. Moosavienia is with the Department of Electrical Engineering, Khajeh Nasiroddin Toosi Industrial University, Seyed Khandan, Tehran, Iran.
K. Mohammadi is with the Department of Electrical Engineering, Iran University of Science and Technology, Narmak, Tehran 16844, Iran.

2 ABFT scheme

ABFT has been suggested to design fault tolerant array processors and systolic array systems. The scheme is capable to detect and sometimes correct errors caused by permanent or transient failures in the system. It was first proposed as a checksum approach for matrix operations [15], [16]. Since then, the technique has been extended to many digital signal processing applications such as Fast Fourier Transform [17], [18], solving linear and partial differential equations [19],[23], digital filters [20] and to protect linear [21] and general multiprocessor systems[22].

Fig.1 shows the basic architecture of an ABFT system. 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.

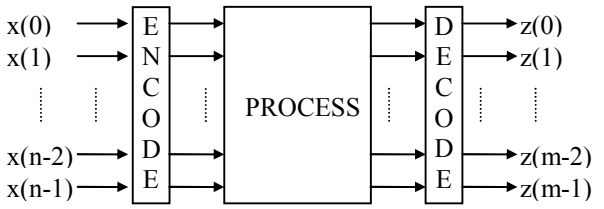

Fig. 1 General architecture of ABFT

The coding algorithm is closely related to the running process and is often defined by real number codes generally of the block types [24]. Systematic codes are of most interest because the fault detection scheme can be superimposed on the original process box with the least changes in the algorithm and architecture.

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

2.1 Convolutional codes

A convolutional encoder, processes data stream sequentially and for every k information symbols 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 decoder is called constraint length. This code has been used as a suitable mechanism in data communication for many years [25]. Although they are basically designed to protect data streams on finite fields, but researches on infinite fields is also reported [26]. We consider only systematic forms of convolutional codes because the normal operation of Process block is not altered and there is no need to decoding for obtaining true outputs. In addition systematic convolutional codes are proved to be noncatastrophic.

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

$$G=\begin{bmatrix} IP_0 & 0P_1 & 0P_2 & \dots & 0P_m & & \\ & IP_0 & 0P_1 & \dots & 0P_{m-1} & 0P_m & \\ & & IP_0 & \dots & 0P_{m-2} & 0P_{m-1} & \dots \\ & & & & \dots & & \\ & & & & & \dots & \dots \end{bmatrix} \quad (1)$$

where *I* and *0* are identity and all zero k×k matrixes respectively [32] and P_i with *i*= 0 to m is a k×(n-k) matrix whose entries are :

$$P_j=\begin{bmatrix} g_{1,j}^{(k+1)} & g_{1,j}^{(k+2)} & \dots & g_{1,j}^{(n)} \\ g_{2,j}^{(k+1)} & g_{2,j}^{(k+2)} & \dots & g_{2,j}^{(n)} \\ \dots & \dots & \dots & \dots \\ g_{k,j}^{(k+1)} & g_{k,j}^{(k+2)} & \dots & g_{k,j}^{(n)} \end{bmatrix} \quad (2)$$

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

$$H=\begin{bmatrix} P_0^T I & & & \\ P_1^T 0 & P_0^T I & & \\ \dots & \dots & \dots & \\ P_m^T 0 & P_{m-1}^T 0 & \dots & P_0^T I \\ & P_m^T 0 & \dots & P_1^T 0 \quad \dots \\ & & & \dots \quad \dots \end{bmatrix} \quad (3)$$

I and 0 are identity and all zero (n-k)×(n-k) matrixes respectively. The syndrome equations, denoted by vector S, are given by:

$$S=rH^T=eH^T \quad (4)$$

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

There are three principal ways of decoding convolutional codes, Viterbi decoding, sequential decoding and majority-logic decoding [24]. Viterbi algorithm is an optimal decoding procedure based on 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 still has a delay as long as input data stream. A majority-logic decoder on the other hand has the least performance but it needs one constraint length of code and just one computation per bit for decoding. So it minimizes memory usage and has the highest decoding speed. This paper therefore uses the majority-logic decoding for its convolutional code.

2.2 Self Orthogonal Codes

Majority logic decoding is based on the orthogonal parity-check sums, i.e. the equations relating any syndrome bit or any sum of them 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. 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 value 1. t_{ML} is called the majority-logic correcting capability of the code [24].

To employ the maximum error correcting capability of the code, it must be completely orthogonalizable [24], that is a code in which $J=d_{min}-1$. By definition d_{min} is:

$$d_{min}=\min\{w[v]_m : u_0 \neq 0\} \quad (5)$$

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 the need for adding syndrome bits. Using these codes make an easier implementation of majority-logic decoding.

3 Data distribution in a MLP network

MLP network consists of several cascaded layers of neurons with sigmoid activation functions [27]. The input vector, feeds into each of the first layer neurons, the outputs of this layer feed into each of the second layer neurons and so on, as shown in Fig. 2.

The layers between input and output are called hidden layers. In this paper feed forward I-H-O neural networks are considered. Which H, O and I denote nodes in input layer, hidden layer and output layer respectively.

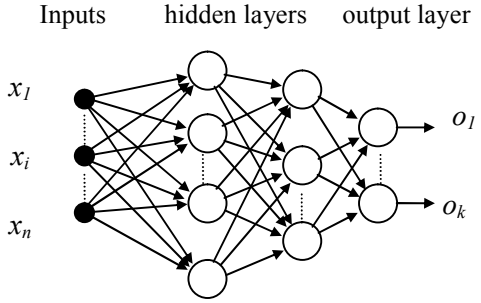

Fig. 2 Architecture of a typical MLP network.

To evaluate the fault tolerance of a MLP network we present two definitions as the following:

Definition 1: $s(w_i)$ is defined as the sensitivity of neural network to weight w_i that is the effect on mean square error (MSE) when w_i is forced to zero.
Sensitivity can be measured by:

$$s(w_i) = |E(W') - E(W)| \tag{6}$$

In which $W=(w_1,...,w_k)$ denotes the vector of all weights of the neural network and W' is the new vector in which w_i is stuck at zero. E(W) is the MSE for weight vector W, over all training set.

Definition 2: The I_{ij}, the information package (IP) of link w_{ij}, is defined as:

$$I_{ij} = oh_i \times w_{ij} \tag{7}$$

Where oh_i is the output of hidden neuron i and w_{ij} is the connecting weight between hidden neuron i and output neuron j.

Most often the nodes are fully connected, i.e., every node in layer l is connected to every node in layer l+1. In this paper we assume input vector as the first layer in the neural network. MLP networks can easily perform Boolean logic operations, pattern recognition, classification and nonlinear function approximation [28]. 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 BP is the most common used learning algorithm that is a steepest descent gradient-based algorithm. In this paper we assume that the activation function of each neuron is a bipolar sigmoid by the following equation:

$$f(s_i) = \frac{1 - \exp(-s_i)}{1 + \exp(-s_i)} \tag{8}$$

$$s_i = \sum_j w_{ij} \times x_i - \theta_i \tag{9}$$

x_j is neuron j's output and θ_i is a bias value for the neuron i. Standard BP algorithm changes w_{ij} in order to reduce the output error, E, defined by:

$$E = \frac{1}{2}\sum_i (t_i - o_i)^2 \tag{10}$$

Where t_j is j'th output target and o_j is the j'th estimated output [29].

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

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{11}$$

η is a positive number called "learning rate" which determines step size in w_{ij} changes. Selecting a suitable η value plays an important role in network learning convergence [30].

Back propagation algorithm says that:

$$\Delta w_{ij} = \eta \delta_i^P o_j^P \tag{12}$$

$$\delta_i^P = (t_i^P - o_i^P).f'(u_i^P) \tag{13}$$

$$\delta_i^P = (\sum_k w_{ki}\delta_k^P).f'(u_i) \tag{14}$$

Where equation (13) is for an output layer and equation (14) is for neurons in hidden layer. $f'()$ is the derivative of the sigmoid and is calculated by:

$$f'(x) = 2f(x)(1 - f(x)) \tag{15}$$

To evaluate the fault tolerant behavior of a MLP trained with standard BP we will continue by an example.

Example 1: A 3-4-1 neural network as in Fig. 3 is trained to approximate a non-linear function defined in equation (16).

$$o = \frac{1}{x + y + z} \tag{16}$$

The training set is $T_L=\{0.1, 0.2,..., 1.1\}$. BP is iterated for 30000 epochs with a learning rate of 0.05. The trained network is then subjected to four stuck-at-0 faults according to each node of hidden layer.

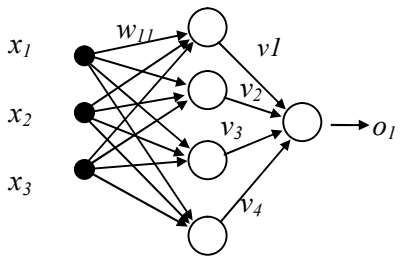

Fig. 3 The 3-4-1 MLP network used in Example 1.

Table 1 shows the sensitivity measures for 10000 input vectors selected randomly from the test set T_T={0.10, 0.11,..., 1.11}.

Table 1 Weight sensitivity for MLP trained by standard BP algorithm

Weight in 1'st layer	sensitivity	Weight in 2'nd layer	sensitivity
w_{11}	0.0758	v_1	1.8808
w_{12}	0.0050	v_2	0.0916
w_{13}	0.0006	v_3	0.0581
w_{14}	0.0012	v_4	0.5503

In this example v_1 -v_4 are the weights in output layer and w_{11}-w_{14} are the weights from input node 1 to all nodes in hidden layer. Clearly the network is not too sensitive to the weights in first layer. However, in the output layer there is a large sensitivity to weight v_1.

It is worth to look at histogram of IPs corresponding to links in output node, shown in Fig. 4. Clearly IP_1 differs significantly from others. It has the biggest peak value and is placed far from origin.

This example shows that the node with maximum sensitivity, v_1 passes most of the information in the neural network architecture, itself.

In other words, if the information distribution becomes uniform, all nodes will have an equal sensitivity.

4 Output error model

Suppose that inputs of a MLP are random variables with a uniform or Gaussian distributed function. Three theorems are presented to model the effect of stuck at 0 faults in a MLP.

Theorem 1: If the inputs of a linear neuron have uniform or Gaussian distributions, then its output will have a Gaussian distribution.
Proof: It is a direct application of central limit theorem [33].♣

Theorem 2: If the inputs of a non-linear neuron with bipolar sigmoid activation function have a uniform or Gaussian distribution, then the distribution of its output is a Gaussian with half mean and variance values of the weighted checksum defined by:

$$S = \sum_{i=1}^{n} w_i x_k \tag{17}$$

Proof: According to theorem 1, S has a Gaussian distribution with mean μ_s and variance σ_s.

$$P_s(S) = \frac{1}{\sqrt{2n}\delta_S} e^{-(\frac{S-\mu_s}{\delta_s})^2} \tag{18}$$

The output y is defined by a bipolar sigmoid function as:

$$y = f(s) = \frac{1-e^{-S}}{1+e^{-S}} \tag{19}$$

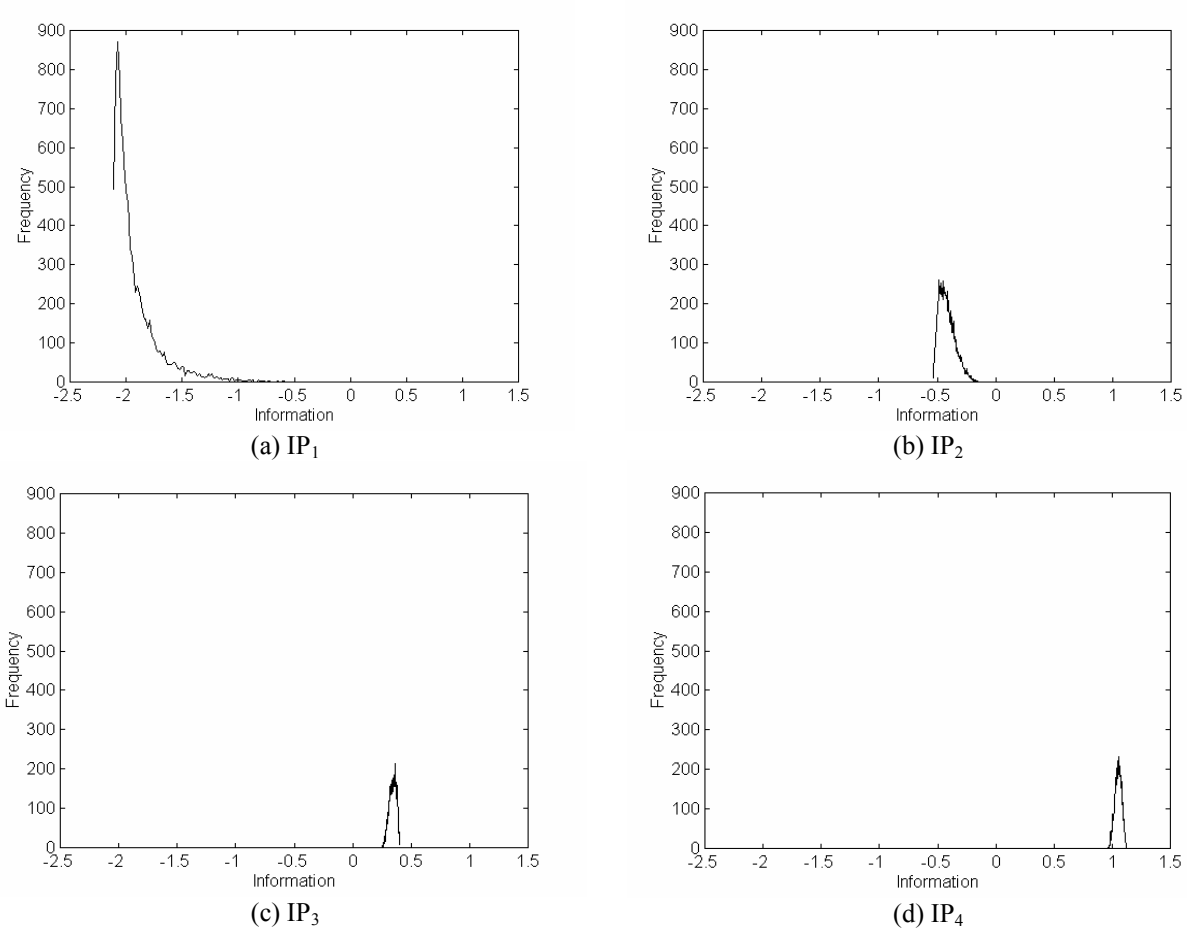

Fig. 4 Distribution function of information packages a) IP_1 b) IP_2 c) IP_3 d) IP_4 in a 3-4-1 MLP trained with conventional BP algorithm

Distribution function of y can be measured as [33]:

$$P_y(y) = |h'(y)| P_S(h(y)) \tag{20}$$

In which h(y) is the inverse of equation (19) defined by:

$$s = h(y) = -\ln\frac{1-y}{1+y} \quad , \quad -1 < y < 1 \tag{21}$$

Differentiating equation (21) result:

$$h'(y) = \frac{2}{1-y^2} \tag{22}$$

By substituting (21) and (22) in (20) results:

$$P_y(y) = \left|\frac{2}{1-y^2}\right| \frac{1}{\sqrt{2\pi}\delta_S} e^{-\frac{(\ln(\frac{1+y}{1-y})-\mu_S)^2}{\delta_S{}^2}} \quad , \quad -1 < y < +1 \tag{23}$$

With the assumption that y is usually around origin, the two following approximations are valid:

$$\frac{2}{1-y^2} \approx 2 \tag{24}$$

$$\begin{aligned} \ln\frac{1+y}{1-y} &= \ln(1+\frac{2y}{1-y}) \\ &\approx \ln(1+2y) \\ &\approx 2y \end{aligned} \tag{25}$$

And substituting equations (25) and (24) in (23), we have:

$$P_y(y) = \frac{1}{\sqrt{2\pi}\frac{\sigma_S}{2}} e^{-\frac{(y-\frac{\mu_S}{s})^2}{(\frac{\sigma_S}{2})^2}} \tag{26}$$

which is a Gaussian with $\mu = \frac{\mu_S}{2}$ and $\sigma = \frac{\sigma_S}{2}$.♣

Theorem 3: If inputs to a neural network are random variables with uniform or Gaussian distributions, error introduced by stuck at 0 faults, obey a Gaussian model too.

Proof: To simplify the proof, suppose that the hidden neurons are non-linear with bipolar sigmoid activation functions but output nodes are linear. Every output node computes a weighted sum on the hidden layer nodes. Now consider oh_k as the output of the hidden node k and w_{ki} as the connecting weights between hidden neuron k and output node i. It is clear that contribution of hidden neuron h to output is equal to $w_{ki*}oh_k$. A stuck at 0 fault in hidden node k, which forces $w_{ki*}oh$ to zero, will introduce an absolute error equal to $w_{ki*}oh_k$. So the output error is proportional to oh_k, which has a Gaussian distribution according to theorem 2. Hence the output error will have a Gaussian distribution too. ♣

For a fixed input pattern the error will choose one of the values from the set $F=\{w_1.oh_1, w_2.oh_2,..., w_n.oh_n\}$. So the expected value and variance of output error are:

$$\mu_f = \frac{1}{N}\sum_{i=1}^{n} w_i.oh_i \tag{27}$$

$$\delta_f = \frac{1}{N}\sum_{i=1}^{n} (w_i.oh_i - \mu_f)^2 \tag{28}$$

In equation (27) the term $\sum w_i.oh_i$ plus the bias value will produce the actual output, which must approximate the target. So μ_f depends on target mean and bias value. On the other hand, equation (28) shows the information package variance. We will show that choosing appropriate weights during the learning phase can reduce σ_f and μ_f.

5 UDBP learning algorithm

Based on the error model introduced in section 4, the UDBP (Uniformly Distributed Back Propagation) algorithm is presented to minimize μ_f and σ_f of output error by distributing information packages uniformly in the neural network architecture. A new error function will be defined as:

$$\tilde{E} = \frac{1}{2}\sum_j (o_j - t_j)^2 + v \times \frac{1}{N}\sum_j \sum_i (oh_i \times w_{ij} - \mu_i)^2 \tag{29}$$

$$\mu_i = \frac{1}{N}\sum_n oh_n \times w_{ni} \tag{30}$$

N denotes the number of hidden layer nodes. The second term in (29) is the total variance of information packages in the neural network. Differentiating (29) in respect to w_{ij} results in:

$$\frac{\partial \tilde{E}}{\partial w_{ij}} = \frac{\partial o_j}{\partial w_{ij}}(o_j - t_j) + \frac{\lambda}{N} oh_i (oh_i w_{ij} - \mu_i) \tag{31}$$

For a linear output the equation is changed to:

$$\frac{\partial \tilde{E}}{\partial w_{ij}} = oh_i [o_j - t_j + \frac{\lambda}{N}(oh_i w_{ij} - \mu_j)] \tag{32}$$

We have introduced a new parameter called λ, which controls the uniformity of information packages. A large value of λ usually tends to a more uniform data distribution in the neural network architecture.

According to these equations the learning is achieved by:

$$w_{ij}(new) = w_{ij}(old) - \mu \frac{\partial \tilde{E}}{\partial w_{ij}} \tag{33}$$

In which, μ is the learning rate parameter.

Similar equations can be written for a bias learning rule:

$$\frac{\partial \tilde{E}}{\partial b_j} = o_j - t_j + \lambda(b_j - \mu_j) \tag{34}$$

And,

$$b_j(new) = b_j(old) - \mu \frac{\partial \tilde{E}}{\partial b_j} \tag{35}$$

The conventional BP algorithm is modified according to equations (34) to (35) for training the output layer weights. The new algorithm, which is called Uniformly Distributed Back Propagation (UDBP), is as the following:

UDBP algorithm:

Step 0: Initialize weights, μ and λ.
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 node receives input signal and broadcast it to hidden layer units.
Step 4: Each hidden node sums its weighted inputs and applies its activation function according to equations (8) and (9).
Step 5: Each output node sums its weighted input signal and produces its output, too.
Step 6:
Step 6-1: For each output node the error information term is computed using equation (31) or (32).
Step 6-2: For each output node the bias gradient term is computed using equation (34).
Step 7: For each hidden node, equations (12) and (13) are computed.
Step 8: For each output and hidden node, weights are updated according to:

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij} \quad \eta$$

Step 9: Test stopping condition.

Example 2: To evaluate UDBP algorithm, the 3-4-1 MLP network in Fig. 3 is trained to approximate the non-linear function of equation (16). The training and testing sets and conditions are the same as Example 1 and λ is 0.8 here. Table-2 shows the sensitivity measures for the trained neural network.

Table 2 Weight sensitivity for 3-4-1 MLP trained by UDBP algorithm

Weight in 1'st layer	sensitivity	Weight in 2'nd layer	sensitivity
w_{11}	0.0343	v_1	0.0359
w_{12}	0.0054	v_2	0.0304
w_{13}	0.0053	v_3	0.0301
w_{14}	0.0051	v_4	0.0289

Comparing table 1 and table 2 it seems that the maximum sensitivity of neural network is reduced from 1.8808 to 0.0359. On the other hand it is clear that the sensitivity of all hidden nodes are approximately equal. Although the UDBP is just applied to output layer, the sensitivity of weights in first layer is also improved.

Fig. 5 shows the distribution function of IP_1 to IP_4 for the network trained by UDBP. Comparing with fig. 4 it is clear that all distributions are moved toward origin and their peak values are very close to each other.

6 Simulation results

Three other algorithms consisting of WRTA (Weight Restricted Training Algorithm), ADP (Addition/Deletion Procedure) [14] and N-FTBP (N Fault Tolerant Back Propagation) [10] are chosen to be compared with UDBP algorithm. The 3-4-1 MLP network used in examples 1 and 2 is trained again for all algorithms. Table 3 shows the measured sensitivity for all hidden layer nodes after completion of training process for unique initial weights.

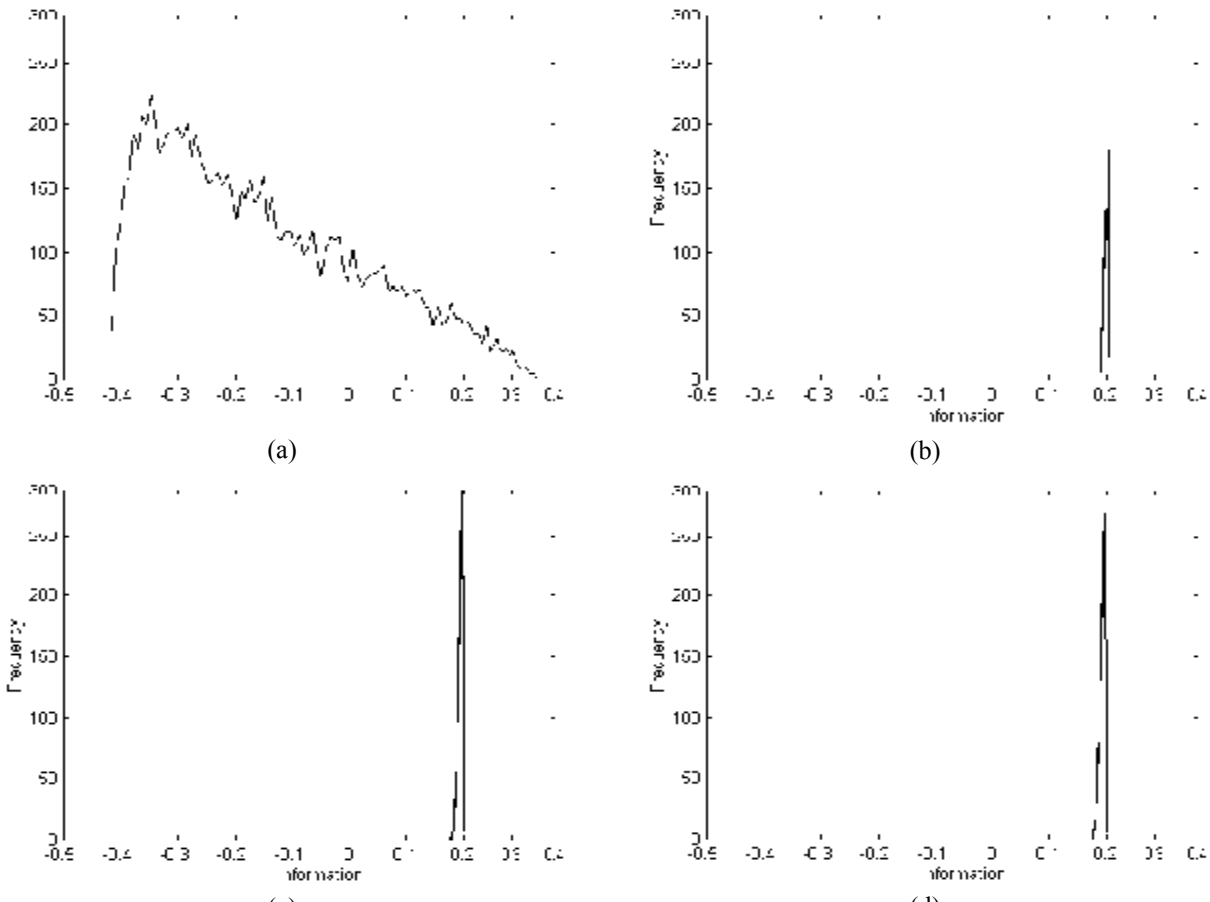

Fig. 5 Distribution function of information packages a) IP_1 b) IP_2 c) IP_3 d) IP_4 In a 3-4-1 MLP network trained with UDBP algorithm

In N-FTBP algorithm one stuck at 0 fault is injected for fifty iterations of BP training algorithm. In WRTA the maximum value of weights for output layer is 1 and for first layer is 5. ADP algorithm performs deletion of the node with least sensitivity and duplication of the one with maximum sensitivity after 100 iteration of BP algorithm. Learning rate is 0.05 for all algorithms. Table-3 shows that UDBP has the most uniform sensitivity among all. The average response of ADP is slightly better. N_FTBP and WRTA show a moderate improvement compared to standard BP. Standard BP has the worst fault tolerance itself.
In another application, a low pass FIR filter [34] with transfer function of:

$$H(z)= -0.0087+0.252z^{-2}+0.5138z^{-3}+0.252z^{-4}-0.0087\,z^{-5} \quad (36)$$

is also approximated with a 6-4-1 MLP. The maximum iteration is 50000 for all mentioned algorithms. Table-4 shows the best computed sensitivity in the output layer obtained for different algorithms.

The 6-4-1 MLP network trained to approximate the FIR filter is a dynamic neural network indeed. Data stream moves from one input node to next one in each time step. This will help the network to produce a more uniform data distribution compared to static neural networks. Table-4 shows that the average sensitivity for a dynamic neural network is less than a static neural network. UDBP has the least sensitivity however the ADP and N_FTBP have moderate responses. WRTA response is near to UDBP.

7 Fault correction using UDBP

Convolutional codes are usually used over the transmission channels, through which both information and parity bits are sent. To achieve fault detection and correction properties of this code in a nonlinear process with the minimum overhead computations, we propose the block diagram in fig 5.
The main architecture is similar to a normal ABFT scheme except of the neural network and delay line 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 then fed to the convolutional encoder to make parity sequence y". On the other hand MLP network is trained by UDBP algorithm to have a direct parity, y', equal to y" in the absence of any noise and faults in system. So the syndrome sequence is a stream of zero or near zero values in normal operation.

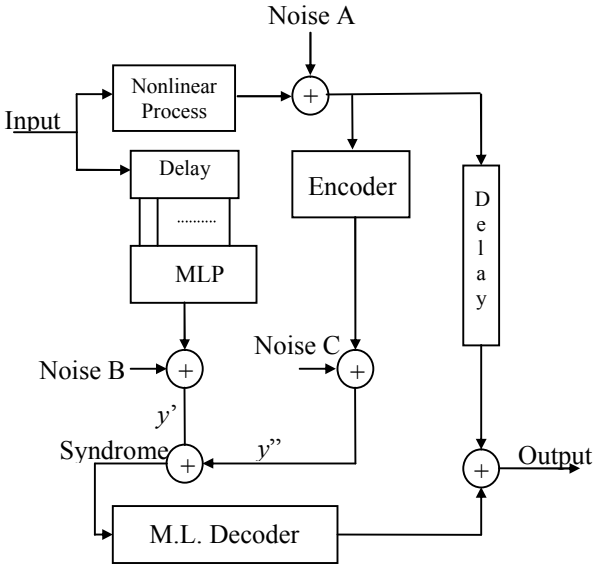

Fig. 5 Block diagram of the neural based ABFT

We have modeled faults in a nonlinear process block with module noise A while the encoder and neural network noises are modeled with modules B and C. Since these two last noises contribute in syndrome additively we can delete one of them without any degradation.

7-1 Example and Simulations

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

$$g_1^{(3)} = 1 + D \quad (37)$$

$$g_2^{(3)} = 1 + D^2 \quad (38)$$

is used to evaluate the error detect ability and correct ability of our proposed method jointed with a MLP with 4 inputs 21 nodes in hidden layer and one output. 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:

Table 3 Sensitivity measures for a 3-4-1 MLP network trained by different algorithms

Algorithm	Sensitivity of nodes in hidden layer				
	v_1	v_2	V_3	v_4	Average
BP	1.8808	0.0916	0.0481	0.5503	0.6452
UDBP	0.0359	0.0304	0.0301	0.0289	0.0313
WRTA	0.2100	0.0867	0.0722	0.0734	0.1150
N-FTBP	0.0111	0.0014	0.0002	0.4205	0.1083
ADP	0.0367	0.0193	0.0375	0.0276	0.0302

Table 4 Sensitivity measures for a 6-4-1 MLP trained by different algorithms

Algorithm	Sensitivity of nodes in hidden layer				
	v_1	v_2	v_3	v_4	Average
BP	0.0003	0.0048	0.0351	0.0014	0.0104
UDBP	0.0032	0.0051	0.0052	0.0046	0.0045
WRTA	0.0060	0.0059	0.0059	0.0056	0.0059
N-FTBP	0.0186	0.0129	0.0002	0.0003	0.0080
ADDROP	0.0010	0.0191	0.0125	0.0010	0.0083

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ & & & 1 & 0 & 1 & 0 & 0 & 1 \\ & & & & 1 & 1 & 0 & 0 & 0 \\ & & & & & & 1 & 0 & 1 \\ & & & & & & & 1 & 1 \end{bmatrix} \quad (39)$$

And according to equation 2-3 the parity check matrix for the first constraint length is:

$$H = \begin{bmatrix} 1 & 1 & 1 & & & & & & \\ 1 & 0 & 0 & 1 & 1 & 1 & & & \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (40)$$

We have two parity triangles for each generator as:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ 1 & 1 & \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} e_0^{(1)} \\ e_1^{(1)} \\ e_2^{(1)} \end{bmatrix} + \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_0^{(2)} \\ e_1^{(2)} \\ e_2^{(2)} \end{bmatrix} \quad (41)$$

It is clear that the code is self orthogonal and we can form a set of two orthogonal check sums on the information error bit, hence, $t_{ML}=1$ and the code can correct single faults in each constraint length of code which is three here. Fig 6a shows the main process which is a two input SINE block. The outputs y_1 and y_2 are subjected to single s-at-0 faults modeled with noise modules A1 and A2. The faulty outputs now shown with b_1 and b_2 are then fed to convolutional encoder as in fig 6b. The generated code stream, y", is compared with MLP output y' as shown in fig 6c. The majority logic now produces two error signals which are fed back to delayed output streams in fig 6a and correct outputs. y_1 and y_2 are corrected outputs which their validity is governed by majority logic decoding rule.

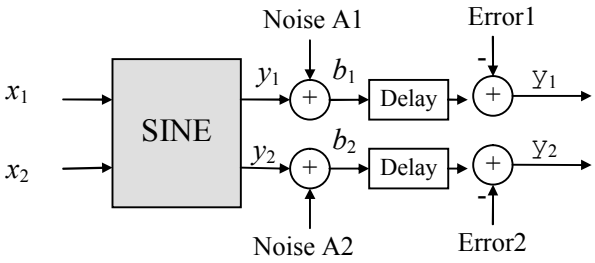

Fig. 6c SINE block input -output

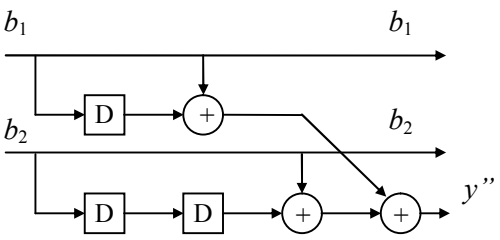

Fig. 6a Systematic (3, 2, 2) convolutional encoder

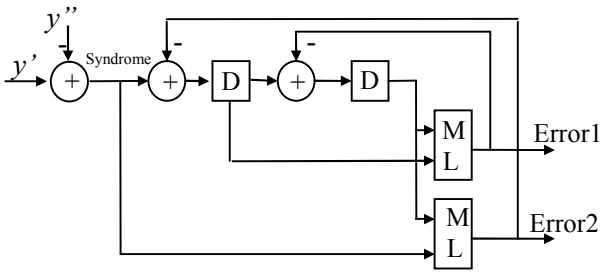

Fig. 6a Complete Majority Logic Decoder for used in this example - ML is the Majority Logic gate

The MLP inputs are selected according to equations (1) and (2). The network is then trained in the absence of any noise so that it's output approximates the SINE block function followed by the (3,2,2) convolution encoder.

Fig. 7 shows the simulation results after 50000 training iterations. In fig-7a the syndrome sequence in the absence of any faults is shown, which is due to neural network limited accuracy. This error value in most times is less than 0.005. However there are also few errors as great as 0.01. Therefore we will select a threshold value equal to 0.05 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 fig 6) and at MLP or Encoder output (module noise C or B in fig 5). Fig-7b shows the syndrome sequence for faults in first output of SINE block, y_1. It is clear that there are two equal nonzero values after each fault occurrence. In fig-7c 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. Fig-6d shows on the other hand, the syndrome for faults in encoder and neural net blocks. It is clear that the syndrome contains single nonzero values for injected faults. Therefore we have three distinct sequences for each fault source which is benefit of convolutional coding used. If there is only a single fault in every constraint length (3 in this case) the majority logic decoding can correct the fault.

8 Conclusions

In this paper we first showed that standard BP algorithm can not yield to a uniform data distribution over the neural network architecture. A measure of sensitivity defined to evaluate fault tolerance of neural network and then we showed that the sensitivity of a link is closely related to the amount of information passes through it. Based on the assumption of using input variables with uniform or Gaussian distribution functions, we proved that the distribution of output error caused by stuck at 0 faults in a MLP network is approximately a Gaussian too. UDBP algorithm then introduced to minimize mean and variance of the output error. Simulation results show that UDBP has the least sensitivity and the highest fault tolerance among other algorithms such as WRTA, N-FTBP and ADP. UDBP has just one extra parameter compared to standard BP. It requires three extra multiplications and two extra additions in each iteration compared to BP.

Then we coupled a MLP neural network trained with UDBP algorithm to a convolutional encoder in an ABFT scheme to demonstrate the feasibility and

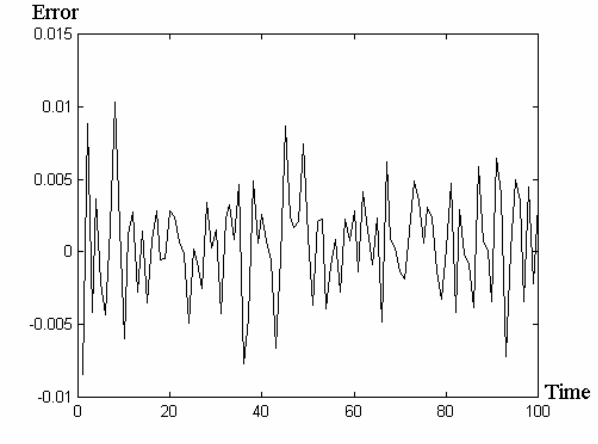

Fig. 7a Syndrome sequence for no fault condition

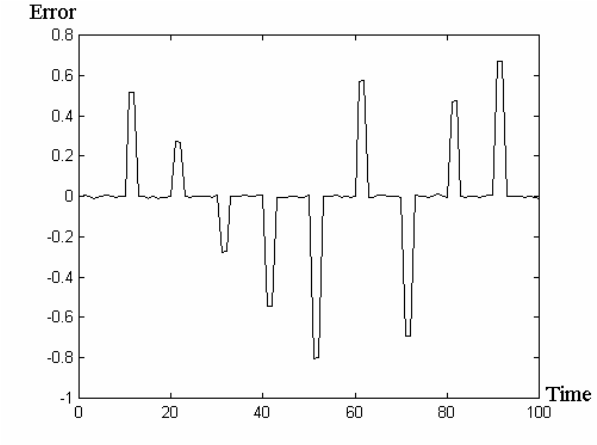

Fig. 7b Syndrome sequence for faults due to Noise source $\bar{A1}$

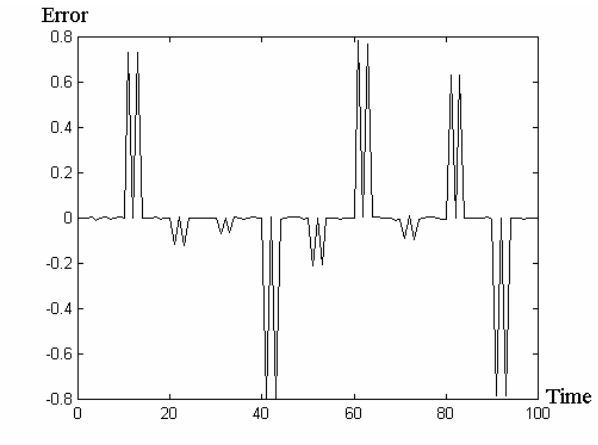

Fig. 7c Syndrome sequence for faults due to Noise source $\bar{A2}$

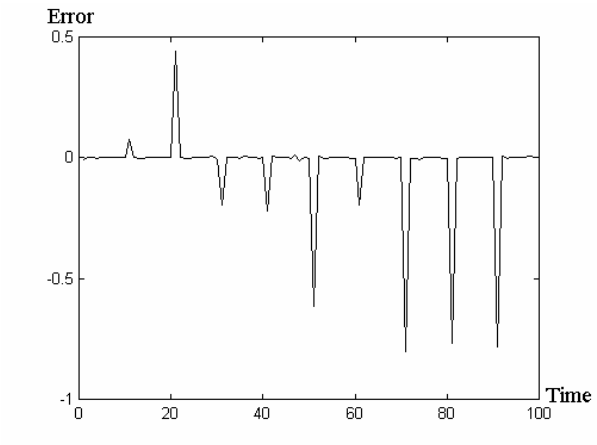

Fig. 7d Syndrome sequence for faults due to Noise sources $\bar{C}$ or B

expandability obtained for fault detection and correction in nonlinear block processes. The trained neural network can itself tolerate single faults, and the used majority logic gates are very simple. So unlike the other ABFT techniques there is no need to apply extra hardware or software to protect these additional blocks. In addition, neural network learn ability permits to change process block functionality without much consideration. Which can not be obtained through conventional ABFT techniques?

9 References

[1] Zoran Vukic, Dean Pavlekovic, H.Ozbolt "Rudder Servo_system Fault Diagnosis Using ",ocean'98 conference Proceedings,Vol.1, Pages 538-543, IEEE 1998.

[2] A. Berneiri, M. D'Apuzzo, L. Sansone, "A Neural Network Approach For Identification And Fault Diagnosis on Dynamic Systems", Instrumentation and Measurment Technology Conference, May 1993, Pages 564-569.

[3] Ching-Yu Tyan and Paul P.Wang,D.R.Bahler, "Neural Fault Diagnosis and Fuzzy Fault Control for a Complex Linear Dynamic System", Fuzzy systems 1995. Int. Joint conf. on the Fourth IEEE International conf. on Fuzzy systems, 1995.

[4] Thitipong Tanprasert, Chularat Tanparasert, "Probing Technique for Neural Net Fault Detection" ,IEEE 1996.

[5] C. Khunasaraphan, K. Vanapipat and C. Lursinsap, "Weight Shifting Techniques for Self-Recovery Neural Networks", IEEE Trans. on Neural Networks Vol. 5, No. 4, July 1994.

[6] B.W.Johnson, Design and Analysis of Fault Tolerant Digital Systems, Addison Wesley Pub. 1989.

[7] Shalini Yajnik and Niraj Jha, "Analysis and Randomized Design of Algorithm Based Fault Tolerant Multiprocessor System Under an Extended Model", IEEE Tran. on Parallel and Distributed Systems, Vol. 8 No. 7 July 1997.

[8] Yasuyuki Taniguchi, Naotake Kamiura, "Activation Function Manipulation for Fault Tolerant Feed forward Neural Networks" , Proceeding of the eight Asian Test Symposium, IEEE 1998.

[9] Alan F. Murrayand Peter J. Edwards, "Synaptic Weight Noise During Multilayer Perceptron Training: Fault Tolerance and

Training Improvement “, IEEE Trans. on Neural Networks Vol. 4, no 4, July 1993.

[10] Takhiro Ito and Itsuo Takanami, “On fault Injection Approaches for Fault Tolerance of Feed forward Neural Networks”, Proceedings of Sixth Asian Test Symposium, Pages 88-93, 1997.

[11] Chalapathy Neti, Michele H. Schneider and Eric D. Young, “Maximally Fault Tolerant Neural Networks”, IEEE Transaction on Neural Networks Vol. 3 No 1, Jan 1992.

[12] Tao Zhang, Dongcheng Hu, Shiyuan Yang, “Fault Tolerant Analysis of Feedback Neural Networks with Threshold Neurons”, Proceeding of the eight Asian Test Symposium, IEEE 1998.

[13] Ching Tai Chiu, Kishan Mehrotra, Chiukuri K. Mohan and Sanjay Ranka, “Training Techniques to Obtain Fault Tolerant Neural Networks", 24’th international symposium on fault tolerant computing, 1994.

[14] O.J.Kwon and S.Y.Bang, “Design of Fault Tolerant multilayer Perceptron with desired level of robustness”, electronic letters Vol. 33 No. 12 1997.

[15] Cynthia J.Anfinson, “A linear Algebraic Model of Algorithm Based Fault Tolerance ”, IEEE Transaction on Computers, Dec 1988, Vol 37 12, Pages 1599-1604.

[16] V.S.S.Nair ,and J. A. Abraham,” Real-Number Codes for Fault Tolerant Matrix Operations On Processor Arrays “,IEEE Trans. on Computers, Vol. 39 No 4 April 1990.

[17] D. L. Tao and C. R. Hartmann, “A novel Concurrent Error Detection Scheme for FFT Networks”, IEEE transaction on parallel and distributed systems, Vol. 4 No. 2 Feb 1993.

[18] Sying, Tyan Wang, “Algorithm Based Fault Tolerant for FFT Networks”, IEEE transaction on computing Vol. 43 No. 7 July 1994.

[19] K. N. Balasubramanya and K. Bhuvaneswari and C. Siva Ram Murthy, “A New Algorithm Based on Givens Rotations for Solving Linear Equations on Fault Tolerant Mesh Connected Processors”, IEEE trans. on parallel and distributed systems, Vol. 9 No. 8 August 1998.

[20] G. Robert Redinbo, and B. Zagar ,” Modifying Real Convolutional Codes for Protecting Digital Filtering Systems “ , IEEE Trans. on Information Theory ,Vol. 39 No 2 March 1993.

[21] Jan-Lung Sung and G. Robert Redinbo, “Algorithm Based Fault Tolerant Synthesis for Linear Operations”, IEEE transaction on computers Vol. 45 No. 4 April 1996.

[22] D. M. Blough and Andrzej Pelc, “Almost Certain Fault Diagnosis Through Algorithm Based Fault Tolerance”, IEEE trans. on Parallel and Distributed Systems, Vol. 5 No. 5 May 1994.

[23] Amber Roy Chowdhury, Prithviraj Banerjee, “Algorithm Based Fault Location and Recovery for Matrix Computations on Multiprocessor Systems”, IEEE Trans. on Computers ,Vol. 45 No 11 Nov. 1996.

[24] John Baylis ,Error- Correcting Codes: A Mathematical Introduction, Chapman and Hall LTD ,1998.

[25] Bernard Sklar, “A Tutorial On Convolutional Coding For M-Ary Signals, Trellis Coded Modulation”,Military Communications Conference, 1988, Pages 637-645.

[26] G. Robert Redinbo and Bernhard Zagar, “Modifying Real Convolutional Codes for Protecting Digital Filtering Systems”, IEEE trans. on Information Theory Vol. 39 No. 2 March 1993.

[27] Laurence Fausett, Fundamental of Neural Networks, Printice Hall , 1994.

[28] Hush and Horne, “Progress in Supervised Neural Networks”, IEEE Signal Processing Magazine, Jan 1993.

[29] R. Beale and T. Jackson, Neural Computing: An introduction, Dep. of computer science, university of YORK, IOP Publishing LTD 1990.

[30] A. Adibi, A. Moosavienia, P. Moallem, “Fast Neural Network Learning using a Variable Learning Parameter Value”, 3rd Iranian Con. on Electrical Engineering , 1995.

[31] Amir Moosavienia, Karim Mohammadi ,“An ABFT For FIR Systems Using a Time Delay Neural Network”, Proceeding of 4th International Wireless and Telecommunications symposium , IWTS2000, P 244-248, May 2000.

[32] Andrew J. Viterbi and J. K. Omura, Principles of Digital Communication and Coding, Mc-Grawhill , 2-nd Print 1985.

[33] Robert Grover Brown and Patrick Y.C. Hwang, Introduction to Random Signal and Applied Kalman Filtering, with MATLAB excersixe and solutions,John Wieley & Sons, 1997.

[34] Edward P. Cunningham, Digital Filtering: An Introduction, John Wiley &Sons, 1995.