J.Sci.Soc.Thailand, 24(1998) 11-20

FAULT IMMUNIZATION MODEL FOR ARTIFICIAL NEURAL NETWORKS

CHIDCHANOK LURSINSAP

Department of Mathematics and Computer Science, Chulalongkorn University, Bangkok 10330, Thailand.

(Received August 9, 1997)

ABSTRACT

By injecting some chemical substances to a cell, it is able to enhance the ability of the cell to fight against the intruder. This immunization concept in biological cells has been applied to enhance the fault tolerance capability in a perceptron-like neuron. In this paper, we consider only the case where each neuron separates its input vectors into two classes. We mathematically model the cell immunization in terms of weight vector relocation and propose a polynomial time weight relocating algorithm. This algorithm can be generalized to the case where each neuron separating the input vectors into more than two classes.

1. INTRODUCTION

Artificial neuron networks have been extensively used in various applications. The reliability of the network depends upon how the network is implemented or used. If the network is implemented as a program running on a general computerthe reliability obviously depends upon the design of that computer. On the other hand if the network is directly implemented on a VLSI circuit the reliability strongly depends upon the design of the network. It has been believed that artificial neural networks have an intrinsic capability of fault tolerance on the synaptic links and neurons due to the redundancy.However this belief is gradually diminished as directly and indirectly reported in these papers in the reference list. Currently, designing a reliable neural network to tolerate faults has been an emerging essential issue.

The problem of designing such a network is categorized into three directions based on the solution and the constraints of the problems.These directions are (1) design of a self-detection of faulty network by applying the techniques used in digital circuit design,[1,3] (2) design of a self-recovery network when there are faulty links and faulty neurons,[9] and (3) design of a fault tolerant network by injecting noise into input vectors, perturbing weight, and pruning some links during retraining.[2,5,6,10,11,13] The first and the third directions gain the most interest from the researchers because of the well-established techniques in digital design and optimization. Faults are assumed to be stuck-at-0, stuck-at-1, and stuck-at-random. The stuck-at-0 and stuck-at-1 are mostly and widely assumed due to their simplicity and easiness. Although these proposed techniques work well in most situationsthere are still some possible improvements to enhance the robustness and reliability of the network.

Our direction is based on the concept of biological immunization against any decease prior to its illness. By injecting some chemical substances to a cell, it is able to enhance theability of the cell to fight against the intruder. In case of an artificial neural network, we inject some constants to the elements of each weight vector so that the weight vector is relocated to the appropriate location. A similar concept was reported by Chun and McNamee.[4] Their technique is based on a trail-and-error process. Some small signal perturbations are

repeatedly injected into the links until the network is non-operational. They also injected noise into the input vectors. Our approach differs from this approach. We directly analyze the input space and compute the new appropriate location for the weight vector. In this paper, we focus our problem and solution on a multi-layer perceptron type network.

A neuron having perceptron-liked function classifies its input data space into two classes by using an activation function such as sigmoid function.[8] This function acts as a threshold decision function. The advantage of this type of classification is the function provides a tolerant range for the output value as long as the threshold rule is not violated. Another advantage is this type of network can be used to emulate other networks used to perform afunctional approximation. We will not discuss this emulation issue in this paper. For any neural network, there aretwo possible failures. The first failure is due to the the deviation ofsynaptic weight of each dendrite or link. The second failure is due to the malfunction of a neuron. In this paper we concentrate on failure at each link.

The paper is organized as follows. Section 2 discusses the fault immunization concept. Section 3 illustrates how the immunization algorithm works with some real examples. Section 4 provides some discussion to the related issues of hardware implementation. Section 5 concludes the paper.

2. FAULT TOLERANCE IMMUNIZATION CONCEPT

2.1 General Concept and Algorithms

Let $W_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,n})$ be the weight vector of neuron i, $X_A^\alpha = (x_{A,1}^\alpha, w_{A,2}^\alpha, \ldots, w_{A,n}^\alpha)$ and $X_A^\beta = (x_{B,1}^\beta, w_{B,2}^\beta, \ldots, w_{B,n}^\beta)$, where α and β are input pattern indicies. We first consider the problem of how to immunize a fault tolerance capability of a single neuron with two-class separation and the solution to the problem. Then we discuss how to generalize the solution to the neuron with multiple-class separation. Basically, a neuron i separating its input vectors into two classes, say A and B, acts as a hyperplane locating in between two input vector groups. The location of the hyperplane is captured by the weight vector W_i of the neuron and the classifying decision is based on the value of the dot product between W_i and X^μ, where X^μ possibly means X_A^α or X_B^β. If $W_i \cdot X^\mu$ is greater than a threshold value τ then X^μ is in class A otherwiseit is in class B. Without loss of generality, we let τ equal zero.

When the training is successful or converged the weight vector representing a hyperplane must be located in the empty space between input vectors in class A and class B. All the vectors in class A are on one side of thehyperplane while all vectors in class B are on the other side. Figure 1 illustrates an example in a 2-dimensional space. In this case, there are three separating lines, $l1, l2, l3$. These lines have the same separating ability but different fault tolerance capability. When some elements of the weight vector of a line, say line 1, change their values, the line will be swung or relocated to a new position. It may cause a misclassification to some vectors in either class A or class B. It is obvious from Figure 1 that line 3 has more freedom to swing than lines 1 and 2 while preserving its correct classification. We say that line 3 is more *fault tolerant* than lines 1 and 2. The interesting problem is how to find the location of line 3 when the locations of classes A and B are already known. The problem of fault tolerant immunization can be definedas follows. To formally define the fault immunization problem, we denote the following symbols and their meanings.

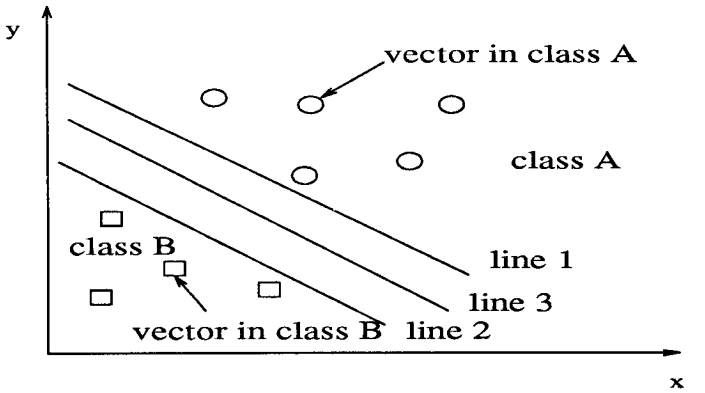

Fig. 1. An example of weight relocation.

1. The trained weight vector of neuron i: $W_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,n})$,
2. The relocated weight vector of neuron i: $W'_i = (w'_{i,1}, w'_{i,2}, \ldots, w'_{i,n})$,
3. Input vectors in class A: $X_A^{\alpha} = (x_{A,1}^{\alpha}, x_{A,2}^{\alpha}, \ldots, x_{A,n}^{\alpha}), 1 \le \alpha \le p$.
4. Input vectors in class B: $X_B^{\beta} = (x_{B,1}^{\beta}, x_{B,2}^{\beta}, \ldots, x_{B,n}^{\beta}), 1 \le \beta \le q$.
5. Trained upper bound constant for $w_{i,j}$: $u_{i,j}$.
6. Trained lower bound constant for $w_{i,j}$: $l_{i,j}$.
7. Relocated upper bound constant for $w'_{i,j}$: $u'_{i,j}$
8. Relocated lower bound constant for $w'_{i,j}$: $l'_{i,j}$

The value of $u_{i,j}, l_{i,j}, u'_{i,j}, and\ l'_{i,j}$ are computedby gradually increasing or decreasing $w_{i,j} and\ w'_{i,j}$, respectively, until a misclassification occurs. Since we focus our pro blem on a 2-classclassification problem then the misclassification is defined as an event when the value of the output of a neuron is less than 0.5 for class B and greater orequal to 0.5 for class A. Therefore, the objective of fault immunization is to find W'_i to minimize $\left|1 - \frac{u'_{i,j} - w'_{i,j}}{w'_{i,j} - l'_{i,j}}\right|, 1 \le j \le n\ and\ 1 \le i \le m,$ to satisfy these conditions:

1. $((w'_{i,1} + u'_{i,1}), \ldots, (w'_{i,n} + u'_{i,n})).(x_{A,1}^{\alpha}, \ldots, x_{A,n}^{\alpha}) \ge 0$, for $1 \le \alpha \le$ p and
2. $((w'_{i,1} + l'_{i,1}), \ldots, (w'_{i,n} + l'_{i,n})).(x_{A,1}^{\alpha}, \ldots, x_{A,n}^{\alpha}) \ge 0$, for $1 \le \alpha \le$ p and
3. $((w'_{i,1} + u'_{i,1}), \ldots, (w'_{i,n} + u'_{i,n})).(x_{B,1}^{\beta}, \ldots, x_{B,n}^{\beta}) < 0$, for $1 \le \beta \le$ q and
4. $((w'_{i,1} + l'_{i,1}), \ldots, (w'_{i,n} + l'_{i,n})).(x_{B,1}^{\beta}, \ldots, x_{B,n}^{\beta}) < 0$, for $1 \le \beta \le$ q.

To achieve the minimum $\left|1 - \frac{u'_{i,j} - w'_{i,j}}{w'_{i,j} - l'_{i,j}}\right|$, first we mustknow the value of each element of W_i and then relocate W_i to obtain W'_i. Thus the following two consequent procedures are needed.

1. Train the network until it converges to a specified limit and obtainthe weight vector W_i for each neuron i.

2. Find W'_i satisfying the above conditions and the objective.

The network can be trained by using any existing learning rule such as Backpropagation .[8,12] The location of W'_i is defined by the shape of the empty space channel lying in between vectors in class A and vectors in class B. The channel is formed by all boundary

vectors in both classes. The problem is how to find these boundary vectors. When the values ofsome $w_{i,j}$'s are forced to either increase or decrease beyond $u_{i,j}$ or $l_{i,j}$ the corresponding neuron will, eventually, misclassify some vectors in both classes. Obviously, these misclassified vectors must lie on the boundary of the channel.

2.2 Finding Boundary Vector Algorithm

In this section, we only focus our discussion on the algorithm of finding boundary vectors based on a single faulty link which may occur to one of $w_{i,j}$'s. Later, we will discuss the reason that this algorithm is still correct when apply to multiple faulty links. Since the size of W'_i therefore we need at least n boundary vectors from either class A or class B to compute the value of each $w'_{i,j}$. The simplest solution to finding these boundary vectors is to gradually increase and decrease each $w_{i,j}$ until a misclassification occurs. The misclassified vectors will be the boundary vectors. Notice that the process of increasing and decreasing the value of $w_{i,j}$ can generate at least one misclassified vector in class A and another misclassified vector in class B. For each neuron i, we find the boundary vectorswith respect to each $w_{i,j}$ as follows. Let X_A^α be the input vector α of class A, X_B^β the input vector β of class B, and W_i the weight vector of neuron i.

Algorithm 1

1. Let $a_{i,j}^\alpha = \dfrac{|W_i \bullet X_A^\alpha|}{x_{A,j}^\alpha}, 1 \le \alpha \le p.$
2. Let $b_{i,j}^\beta = \dfrac{|W_i \bullet X_B^\beta|}{x_{B,j}^\beta}, 1 \le \beta \le q.$
3. Find $\min_\alpha (a_{i,j}^\alpha)$ and $\min_\beta (b_{i,j}^\beta)$.

Theorem 1 *With respect to $w_{i,j}$, an input vector corresponding to $\min_\alpha (a_{i,j}^\alpha)$ is the boundary vector in class A and an input vectorcorresponding to $\min_\beta (b_{i,j}^\beta)$ is boundary vector in class B.*

Proof Each $w_{i,j}$, can be rewritten in this form $(w_{i,j} + \delta_j)$,where δ_j is a constant and is equal to zero in normal situation. Without loss of generality, we only consider the boundary vectors in class A andprove only when $w_{i,j}$ is increased. The proof for the boundary vectors in class B and decreasing $w_{i,j}$ have the similar argument. Therefore the dot product $W_i . X_A^\alpha$ becomes

$$\begin{aligned} W_i . X_A^\alpha &= w_{i,1} x_{A,1}^\alpha + w_{i,2} x_{A,1}^\alpha + \ldots + (w_{i,j} + \delta_j) x_{A,j}^\alpha + \ldots + w_{i,n} x_{A,n}^\alpha \\ &= P + \delta_j x_{A,j}^\alpha \end{aligned}$$

where $P = w_{i,1} x_{A,1}^\alpha + w_{i,2} x_{A,1}^\alpha + \ldots + w_{i,j} x_{A,j}^\alpha + \ldots + w_{i,n} x_{A,n}^\alpha$. If we map the value of the dot product as a point to a real line we can see that the value of P represents the distance of this point from the threshold, which is equal to zero. From the above equation, it can be seen that increasing or decreasing δ_j is equivalent to constantly adding or subtracting $\delta_j x_{A,j}^\alpha$ from $w_{i,j}$. Therefore, the number of times to increase or decrease δ_j until a misclassification occurs can be easily computed by dividing $W_i \bullet X_A^\alpha$ with $x_{A,j}^\alpha$. Hence, those vectors having $\min_\alpha (a_{i,j}^\alpha)$ will be the boundary vectors.

Theorem 1 tells us how to find the boundary vectors in classes A and B. These boundary vectors may not be unique with respect to its $w_{i,j}$. This means that different $w_{i,j}$ and $w_{i,k}$ may generate the same set of boundary vectors. The following collorary states the condition of the uniqueness of the boundary vectors with respect to any $w_{i,j}$ and $w_{i,k}$.

Collorary 1 Vector X_A^α is a boundary vector with respect to $w_{i,j}$ and $w_{i,k}$ if $x_{A,j}^\alpha = x_{A,k}^\alpha$.

Proof If X_A^α is a boundary vector with respect to $w_{i,j}$ and $w_{i,k}$ then we must have $\frac{|W_i \bullet X_A^\mu|}{x_{A,j}^\mu} = \frac{|W_i \bullet X_A^\alpha|}{x_{A,k}^\alpha}$ This implies that $x_{A,j}^\alpha = x_{A,k}^\alpha$.

2.3 Weight Vector Relocating Algorithm

After all the boundary vectors are found, the next step is to relocate the weight vector by using the boundary vectors as the reference points. The new location of the weight vector must satisfy the conditions stated in Section 2.1. The problem of weight vector relocation in this case can be formulated as anoptimization problem with the following cost function

$$E = \sum_{k=1}^{K} (f_{A,k} + f_{B,k})^2 \tag{1}$$

$$= \sum_{k=1}^{K} (W'_i X_A^k + W'_i X_A^k)^2 \tag{2}$$

where $f_{A,k}$ is the dot product of the new weight vector and the boundary vector pair k in class A, $f_{B,k}$ is the dot product of the new weight vector and the boundary vector pair k in class B, and K is the number of boundary vector pairs. Here, we consider the bias term as one element in the weight vector with its coefficient of one. Since the weight vector represents a hyperplane in a space of n-1 dimensions and there are n unknown w_i's, it is impossible to solve for all w_i's. Hence we set the value of w_1 to one. This setting can be viewed as normalizing the value of all w_i's by the value of w_1. It will not effect the solution to this problem. The minimization of the cost function is achieved by differentiating it with respect to each element of W'_i and set the result to zero. Then we solve the following set of these linear equations to find each $w'_{i,j}$.

$$\frac{\partial E}{\partial w'_{i,2}} = 0 \tag{3}$$

$$\frac{\partial E}{\partial w'_{i,3}} = 0 \tag{4}$$

$$\vdots \tag{5}$$

$$\frac{\partial E}{\partial w'_{i,n}} = 0 \tag{6}$$

The cost function, E, depends directly upon how we define the boundary vector pairs. The boundary vector pair k consists of two nearest boundary vectors which one vector is from class A and the other is from class B. This pair is used to define the distance of the channel between classes A and B for relocating the weight vector. Let $d(V_j, V_k)$ be the Euclidean distance between V_j and V_k. The boundary vector pair can be found by the following algorithm.

Algorithm 2

1. For each vector X_A^i, find a vector X_B^k such that $d(X_A^i, X_B^k)$ is minimum. Vectors X_A^i and X_B^k are boundary vector pair.

2. For each vector X_B^l, find a vector X_A^m such that $d(X_B^l, X_A^m)$ is minimum. Vectors X_B^l and X_A^m are boundary vector pair.

An example of boundary vector pairs is shown in Figure 2. There are two vectors in class A and three vectors in class B. The boundary vector pairs are {(1,3),(2,4),(5,2)}.

2.4 Tolerance Measure

We define the following measure to evaluate the tolerance of $w_{i,j}$ and $w'_{i,j}$ computed from the algorithm. All the symbols are already defined in Section 2.1.

$$T_{i,j} = \left| 1 - \frac{w_{i,j} - l_{i,j}}{u_{i,j} - w_{i,j}} \right| \tag{7}$$

$$T'_{i,j} = \left| 1 - \frac{w'_{i,j} - l'_{i,j}}{u'_{i,j} - w'_{i,j}} \right| \tag{8}$$

The minimum values of $T_{i,j}$ and $T'_{i,j}$ are equal to 0. This implies that if the weight vector is relocated at the best position in between the boundary of classes A and B then the ratios between $\frac{w_{i,j} - l_{i,j}}{u_{i,j} - w_{i,j}}$ and $\frac{w'_{i,j} - l'_{i,j}}{u'_{i,j} - w'_{i,j}}$ will be equal to one. Weight $w_{i,j}$ achieves the maximum tolerance when $T_{i,j}$ is zero. The value of $w_{i,j}$ can swing towards either the lower bound or the upper bound. The swinging direction and the distance can be easily captured by ignoring the absolute operation from the measure of $T_{i,j}$ and $T'_{i,j}$. If $T_{i,j}$ ($T'_{i,j}$) is greater than zero it indicates that the lower bound distance is less than the upper bound distance. On the other hand, if $T_{i,j}$ ($T'_{i,j}$) is less than zero it indicates the lower bound distance is greater than the upper bound distance.

3. EXAMPLES

To illustrate how the immunization technique works, we will apply it to two examples. The first example concerns the data set in a 2-dimensional binary space. The second example focuses the data set in a 2-dimensional real space. In both examples, only one neuron is used to classify a given data set. We name each weight element w_1, w_2, and w_3.

In the first example, the data set consists of four input vectors with their targets which are {(0,0,0), (1,0,0), (1,1,1), (0,1,0)}. The first two elements are input elements while the last element is the target. We assign class A to the data vector having target 1 and class B to the data vectors having target 0. After training, we obtain the following weight vector w_1 =10.245, w_2 = 10.245, and w_3 = -15.452. The boundary vector in class A is {(1,1)} and the boundary vectors in class B are {(1,0),(0,1)}. These vectors are, then, used to find the boundary vector pairs for relocating the weight vector. In class A, vector (1,1) forms boundary vector pairs with boundary vectors in class B which are {(1,1),(1,0)} and {(1,1)(0,1)}. In class B, both vectors (1,0) and (0,1) form boundary vector pairs with vector (1,1) in class A. Therefore, we only consider these boundary vector pairs {(1,1),(1,0)} and {(1,1),(0,1)} to solve for each new w'_2, and w'_3. The value of w'_1 is set to one. The error function is defined as follows.

$$E = (1+w'_2+w'_3+1+w'_3)^2 + (1+w'_2+w'_3+w'_2+w'_3)^2 \tag{9}$$

Table 1. Tolerance measure of the original weight vector.

w'_i	*Lower Bound*	*Value of* $w'_{i,j}$	*Upper Bound*	*Tolerance Measure*
w'_1	5.094	10.245	15.498	0.019
w'_2	5.094	10.245	15.498	0.019
w'_3	-20.705	-15.452	-10.145	0.0102

Table 2. Tolerance measure of the relocated weight vector.

w_i	*Lower Bound*	*Value of* $w_{i,j}$	*Upper Bound*	*Tolerance Measure*
w_1	0.472	1.0	1.528	0.0
w_2	0.472	1.0	1.528	0.0
w_3	-2.05	-1.5	-0.95	0.0

The value of w'_2 and w'_3 are computed by solving these two linear equations

$$\frac{\partial E}{\partial w'_2} = 0 \tag{10}$$

$$\frac{\partial E}{\partial w'_3} = 0 \tag{11}$$

(12)

which give the new weight vector (1.0, 1.0, -1.5). A threshold value of 0.5 is used for finding the lower and upper bounds of $w_{i,j}$ and $w'_{i,j}$. Tables 1 and 2 summarize the tolerance measures of w_1, w_2, w_3, w'_1, w'_2, and w'_3.

In the second example, the data set consists data in a 2-dimensional real space which are {(0.02,0.06,0), (0.04,0.07,0), (0.05,0.05,0), (0.07,0.05,0), (0.08,0.04,0), (0.06,0.09,1), (0.07,0.07,1), (0.08,0.09,1), (0.09,0.06,1), (0.1,0.05,1), (0.11,0.06,1)}. The vectors are distributed in a zigzag shape. The first two elements are the input elements and the last element is the target. After training we obtain weight vector (217.242,288.607,-32.561). The weight vector obtained from the cost function is (1,1.5,-0.16). In class *A* (target 1), the boundary vectors are (0.07,0.07) and (0.1,0.05). In class *B* (target 0), the boundary vectors are (0.04,0.07) and (0.07, 0.05). Tables 3 and 4 summarize the tolerance measures obtained from training and cost function.

Table 3. Tolerance measure of the original weight vector.

w_i	*Lower Bound*	*Value of* $w_{i,j}$	*Upper Bound*	*Tolerance Measure*
w_1	176.487	217.242	262.995	0.109
w_2	216.597	288.607	347.26	0.228
w_3	-36.341	-32.561	-28.781	0.0

Table 4. Tolerance measure of the original weight vector.

w'_i	*Lower Bound*	*Value of* $w'_{i,j}$	*Upper Bound*	*Tolerance Measure*
w'_1	0.769	1.0	1.25	0.076
w'_2	1.175	1.5	1.8	0.083
w'_3	-0.19	-0.16	-0.13	0.0

4. DISCUSSIONS

Several related issues will be discussed in this section. These issues include (a) the technique of how to further increase the tolerance interval based on the hardware implementation and (b) set of boundary vectors obtained from Algorithm 1.

4.1 Tolerance Interval

The tolerance capability obtained from the technique previously discussed can be further enhanced by carefully considering the tolerance measure of each $w_{i,j}$. Before any further discussion, we define the following terms.

Definition 1 *An upper tolerance interval,* $U_{i,j}$, of $w_{i,j}$ is $|u_{i,j} - w_{i,j}|$.

Definition 2 *An lower tolerance interval,* $L_{i,j}$, of $w_{i,j}$ is $|w_{i,j} - l_{i,j}|$.

The tolerance measure discussed in the previous section indicates how symmetrical each $w_{i,j}$ can deviate either increasingly or decreasingly from this current location. Although this measure does not provide us any information regarding the deviating distance from the current location it gives us freedom to scale up the tolerance intervals $U_{i,j}$ and $L_{i,j}$ to a certain limit as long as the tolerance ratio is constant. This means that the value of each $w_{i,j}$ is also increased by the same scaling factor without effecting the classification capability of the neuron i. We prove this fact in the following theorem. Let f_i be the dot product of weight vector W_i with input vector X and $\tilde{f}_i$ equal σf_i, where $\sigma > 0$ is a scaling factor.

Theorem 2 *Both* f_i *and* $\tilde{f}_i$ *correctly classify the input data into classes A and B.*

Proof The classification is defined by the following rules. The data are in class A *if* $f_i \geq 0$ and in class B if $f_i < 0$. Multiply both sides of the inequality by σ we still obtain $\tilde{f}_i \geq 0$ for data in class A and $\tilde{f}_i < 0$ for data in class B.

Generally, a neuron is always implemented by a digital circuit[7]. In this case, the limit of tolerance interval is constrained by the size of register used to store the value of $w_{i,j}$. We assume that for any neuron, every $w_{i,j}$ uses register of the same size. Let r be the size of a register. Hence, the maximum value to be stored in this register is $2^r - 1$. There are two possible cases. The first case is when $2^r - 1$ is less than $\max_j(u_{i,j})$ and the second is when $\max_j(u_{i,j})$ is larger than $2^r - 1$. $u_{i,j}$ is the upper bound of $w_{i,j}$.

For the first case, we scale the value of $\max_j(u_{i,j})$ down until it is equal to $2^r - 1$. Thus, the scaling factor in this case is equal to $\frac{\max_j(u_{i,j})}{2^r - 1}$. Similarly, we scale $\max_j(u_{i,j})$ upuntil it is equal to $2^r - 1$ for the second case. Hence the scaling factorin this case is $\frac{2^r - 1}{\max_j(u_{i,j})}$. The scaling factor is then used to scale the value of $w_{i,j}$. Tables 5 and 6 summarize the tolerance intervals in both cases after scaling.

Table 5. Lower bound tolerance intervals in each case.

Cases	*Scaling Factor*	*Original* $L_{i,j}$	*New* $L_{i,j}$
$2^r - 1 < \max_j(u_{i,j})$	$\frac{\max_j(u_{i,j})}{2^r - 1}$	$w_{i,j} - l_{i,j}$	$\frac{\max_j(u_{i,j})}{2^r - 1}(w_{i,j} - l_{i,j})$
$\max_j(u_{i,j}) < 2^r - 1$	$\frac{2^r - 1}{\max_j(u_{i,j})}$	$w_{i,j} - l_{i,j}$	$\frac{2^r - 1}{\max_j(u_{i,j})}(w_{i,j} - l_{i,j})$

Table 6. Upper bound tolerance intervals in each case.

Cases	*Scaling Factor*	*Original* $U_{i,j}$	*New* $U_{i,j}$
$2^r - 1 < \max_j(u_{i,j})$	$\frac{\max_j(u_{i,j})}{2^r - 1}$	$u_{i,j} - w_{i,j}$	$\frac{\max_j(u_{i,j})}{2^r - 1}(u_{i,j} - w_{i,j})$
$\max_j(u_{i,j}) < 2^r - 1$	$\frac{2^r - 1}{\max_j(u_{i,j})}$	$u_{i,j} - w_{i,j}$	$\frac{2^r - 1}{\max_j(u_{i,j})}(u_{i,j} - w_{i,j})$

4.2 Correctness of Algorithm 1

Algorithm 1 finds the boundary vectors by separately considering each individual $w_{i,j}$. The problem that we are interested in is whether these boundary vectors are different from the boundary vectors obtained by either simultaneously increasing or decreasing all $w_{i,j}$'s. The answer is no. The difference between considering each individual $w_{i,j}$ and simultaneously considering all $w_{i,j}$'s is the first one is based on the assumption of a single fault while the second one is based on multiple faults. The following theorem verifies the answer.

Theorem 3 *The boundary vector sets in cases of single and multiple faults are the same.*

Proof We prove only the case of increasing $w_{i,j}$. The case of decreasing $w_{i,j}$ has the same argument. Without loss of generality, suppose that $w_{i,j}$ is the element that we consider and the input vector is $(x_1, x_2, \ldots x_n)$. Let δ be a small deviation constant for $w_{i,j}$ in a single fault case and δ_k a small deviation constant for $w_{i,k}$ in a multiple fault case. If the boundary vectors in the case of single fault are different from those in the case of multiple fault then the dot product of the single faulty weightvector with the input vector is larger the dot product of the multiple faulty weight vector with the input vector. Therefore we have

$$w_{i,1}x_2 + \ldots + (w_{i,j} + \delta)x_j + \ldots + w_{i,n}x_n \geq (w_{i,1} + \delta_1)x_1 + \ldots + (w_{i,n} + \delta_n)x_n$$

$$\delta x_j \geq \delta_1 x_1 + \delta_2 x_2 + \ldots + \delta_n x_n$$

Under the same environment, δ and δ_k must be monotonically either increased or decreased. Since the smallest values of δ and δ_k are the same, for every k, we divide both sides by δ. So the inequality becomes

$$x_1 \geq x_1 + x_2 + \ldots + x_n$$

It can be seen that this inequality is not correct because each x_j spositive. This means that when gradually increasing or decreasing δ_i of each $w_{i,j}$ in case of multiple fault the weight vector will eventually touch the same boundary vectors.

5. CONCLUSION

A new technique to further enhance the fault tolerance capability of a neural network called *fault immunization* is proposed. The technique can well support the current fault tolerance techniques. We have demonstrated that if a weight vector is appropriately relocated it will obviously improve thefault tolerance of the network. We also show that it would be better to make the value of each element of the weight vector as large as possible so that it can tolerate the deviation due to faults. Although this paper discusses how to immunize a neuron performing classification it is possible to extend it to a neuron performing functional approximation.

ACKNOWLEDGEMENT

I gratefully thank Thailand Research Fund for kindly support this work under contract RSA 11/2538 and Research Division of Chulalongkorn University for supporting the computing tools.

REFERENCES

1. C. Chen, L. Chu, D. Saab, "Reconfiguration Fault Tolerant Neural Network", International Joint Conference on Neural Networks, p.547-52, vol. 2, 1992.
2. C. Chiu, K. Mehrotra, C.K. Mohan, and S. Ranka, "Training Techniques to Obtain Fault-Tolerant Neural Networks", The 24th International Symposium onFault Tolerant Computing, p.360-9, 1994.
3. L. Chu and B. Wah, "Fault Tolerant Neural Networks with Hybrid Redundancy", International Joint Conference on Neural Networks, 1990, pp. II-639-II-649.
4. Chun and L.P. McNamee, "Immunization of Neural Networks Against Hardware Faults", *IEEE International Symposium on Circuits and Systems*, pp. 714-718,1990.
5. R. Clay and C. Sequin, "Limiting Fault-Induced Output Errors in ANN", Proceedings of International Joint Conference on Neural Networks, 1991,Vol. 2, pp. II-A-965.
6. H. Elsimary, S. Mashali, A. Darwish, and S. Shaheen, "Performance Evaluation of Novel Fault Tolerance Training Algorithm", IEEE International Conference on Neural Networks, p.856-61, vol. 2, 1994.
7. W. Fornaciari and F. Salice, "A New Architecture for the Automatic Design of Custom Digital Neural Networks", *IEEE on VLSI Systems*, Dec. 1995, pp. 502-506.
8. J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of NeuralComputation*, Addison-Wesley Publish Company, 1991.
9. 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.
10. C. Lin and I. Wu, "Maximizing Fault Tolerance in Multilayer Neural Networks", *IEEE International Conference on Neural Networks*, p.419-24, vol. 1, 1994.
11. C. Neti, M.H. Schneider, and E.D. Young, "Maximally Fault Tolerant Neural networks", *IEEE Trans. on Neural Networks*, pp. 14-32, 1992.
12. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation", in *Parallel Distributed Processing*, The MIT Press, 1986.
13. P. Ruzicka, "Learning Neural Networks with Respect to Tolerances to Weight Error", *IEEE Trans. on Neural Networks*, vol. 40, May 1993.