# แบบจำลองและขั้นตอนวิธีการสร้างภูมิคุ้มกันความผิดพร่องสำหรับโครงข่ายประสาทเทียมชนิดจัดกลุ่มเอง

นาง รัชนีวรรณ ตาพุมาศสวัสดิ์



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2543
ISBN 974-346-471-9

# FAULT IMMUNIZATION MODEL AND ALGORITHM FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK

**Mrs. Rachaneewan Talumassawatdi**



**A Thesis Submitted in Partial Fulfillment of the Requirements**
**for the Degree of Master of Science in Computational Science**
**Department of Mathematics**
**Faculty of Science**
**Chulalongkorn University**
**Academic Year 2000**

**ISBN 974-346-471-9**

| | |
|---|---|
| Thesis Title | FAULT IMMUNIZATION MODEL AND ALGORITHM FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK |
| By | Mrs. Rachaneewan Talumassawatdi |
| Department | Mathematics |
| Thesis Advisor | Professor Chidchanok Lursinsap, Ph.D. |

---

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

…..…… Dean of Faculty of Science
(Associate Professor Wanchai Phothiphichitr, Ph.D.)

THESIS COMMITTEE

…..…… Chairperson
(Assistant Professor Peraphon Sophatsathit, Ph.D.)

…..…… Thesis Advisor
(Professor Chidchanok Lursinsap, Ph.D.)

…..…… Member
(Jaruloj Chongstitvatana, Ph.D.)

รัชนีวรรณ ตาฟุมาศสวัสดิ์ : แบบจำลองและขั้นตอนวิธีการสร้างภูมิคุ้มกันความผิดพร่องสำหรับโครงข่ายประสาทเทียมชนิดจัดกลุ่มเอง (FAULT IMMUNIZATION MODEL AND ALGORITHM FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK) อาจารย์ที่ปรึกษา : ศ.ดร. ชิดชนก เหลือสินทรัพย์ ; 50 หน้า ISBN 974-346-471-9

โครงข่ายประสาทเทียมชนิดจัดกลุ่มเอง $(SOM)$ ได้ถูกใช้อย่างกว้างขวางในการแบ่งกลุ่มรูปแบบ การจัดเวคเตอร์เป็นกลุ่ม และการบีบอัดภาพ เราพิจารณาปัญหาของการเพิ่มความแข็งแกร่งของความมั่นคงของโครงข่ายประสาทแบบ $SOM$ โดยวิธีการสร้างภูมิคุ้มกันต่อความผิดพร่องของเส้นเชื่อมโยงประสาทของแต่ละเซลล์ประสาท ซึ่งคล้ายกับแนวคิดการสร้างภูมิคุ้มกันในทางชีววิทยา แทนที่จะมีสมมุติฐานว่า ค่าผิดคงที่ที่ $0$ และค่าผิดคงที่ที่ $1$ ดังเช่นการศึกษาอื่นๆ ก่อนหน้านี้ เราพิจารณากรณีทั่วไปของค่าผิดคงที่ที่ค่า $a$ เมื่อ $a$ คือค่าจำนวนจริง สมมุติฐานอย่างเดียวที่เราพิจารณาก็คือมีเพียงหนึ่งเซลล์ประสาทเท่านั้นที่เสียได้ที่เวลาใดเวลาหนึ่ง ไม่มีข้อจำกัดเกี่ยวกับจำนวนเส้นเชื่อมโยงประสาทที่เสียในแต่ละเซลล์ประสาท กำหนดให้ $w_{i,j}$ เป็นค่าน้ำหนักของเส้นเชื่อมโยงประสาทเส้นที่ $j$ ของเซลล์ประสาท $i$ ซึ่งได้ถูกกำหนดให้หลังจากการแบ่งกลุ่มแบบผู้ชนะเอาหมด $(winner-take-all)$ น้ำหนัก $w_{i,j}$ ถูกสร้างภูมิคุ้มกันโดยการบวกค่าคงที่ $\varepsilon_{i,j}$ ซึ่งเป็นได้ทั้งค่าบวกหรือลบกับ $w_{i,j}$ เซลล์ประสาทบรรลุการมีภูมิคุ้มกันต่อความผิดพร่อง ถ้าค่าของ $w_{i,j}+\varepsilon_{i,j}$ สามารถถูกเพิ่มขึ้นหรือลดลงได้มากที่สุด โดยไม่ทำให้เกิดการแบ่งกลุ่มข้อมูลผิด ดังนั้นปัญหาของการสร้างภูมิคุ้มกันต่อความผิดพร่อง จึงถูกจัดให้เป็นปัญหาของการหาผลลัพธ์ที่ดีที่สุดของค่าของแต่ละ $\varepsilon_{i,j}$ วิธีหาค่า $w_{i,j}+\varepsilon_{i,j}$ และการประยุกต์ใช้วิธีนี้เพื่อเพิ่มความมั่นคงในการส่งภาพที่บีบอัด จึงได้ถูกเสนอแนะขึ้นในวิทยานิพนธ์นี้



ภาควิชา คณิตศาสตร์ ลายมือชื่อนิสิต.....

สาขาวิชา วิทยาการคณนา ลายมือชื่ออาจารย์ที่ปรึกษา.....

ปีการศึกษา 2543 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม.....

RACHANEEWAN TALUMASSAWATDI: FAULT IMMUNIZATION MODEL AND ALGORITHM FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK. THESIS ADVISOR: PROF. CHIDCHANOK LURSINSAP, Ph.D. 50 pp. ISBN 974-346-471-9

Self-Organizing Mapping $(SOM)$ neural network has been widely used in pattern classification, vector quantization, and image compression. We consider the problem of strengthening the reliability of an $SOM$ neural network by applying a fault immunization technique to each neuron synaptic links, which is similar to the concept of biological immunization. Instead of assuming the stuck-at-$0$ and stuck-at-$1$ as in the previous studies, we consider a general case of stuck-at-$a$, where $a$ is a real value. Our only assumption is that only one neuron can be faulty at any time. There is no restriction on the number of faulty links of the neuron. Let $w_{i,j}$ be the weight of synaptic link $j$ of neuron $i$ obtained after the winner-take-all classification. Weight $w_{i,j}$ is immunized by adding a constant $\varepsilon_{i,j}$, either positive or negative, to $w_{i,j}$. A neuron reaches its maximum fault immunization if the value of $w_{i,j}+\varepsilon_{i,j}$ can be either increased or decreased as much as possible without creating any misclassification. Thus, the fault immunization problem is formulated as an optimization problem on finding the value of each $\varepsilon_{i,j}$. A technique to find the value of $w_{i,j}+\varepsilon_{i,j}$ and its application to enhance the transmission reliability in image compression area is proposed in this thesis.



Department Mathematics Student's signature……

Field of study …Computational Science Advisor's signature……

Academic year 2000 Co-advisor's signature……

# Acknowledgements

Although it is not possible to name all those who have been extremely supportive, I would like to acknowledge the support and assistance of several special individuals. Most importantly, I am greatly indebted to my supervisor, Prof. Dr. Chidchanok Lursinsap, at The Advanced Virtual and Intelligent Computing Center (AVIC), for his invaluable advice and great patience in guiding me through my Master's program.

In addition, I would like to acknowledge the advice of Dr. Royol Chitradon, Head of The High Performance Computing Laboratory at NECTEC, who encouraged me to participate in the program. I owe a great deal to Khun Saneh Warit, Former Director of Bureau of the Royal Rainmaking and Agricultural Aviation (BRRAA), who has given me support and encouragement. I would like to thank my superior officers for their permission to take leave, my colleagues at BRRAA, and Assist. Prof. Dr. Pornchai Satarvaha and my friends at Chulalongkorn University, and finally my family especially my only daughter who has been patient with my lack of time for her.

# TABLE OF CONTENTS

**PAGE**

# LISTS OF TABLES

**TABLE** **PAGE**

# LIST OF FIGURES

**FIGURE** **PAGE**

# CHAPTER I

# INTRODUCTION

Self-Organizing Mapping $(SOM)$ neural network $[1]$ is important in analyzing large quantities of data to classify patterns and characteristics $[14]$. With current Very-Large-Scale Integrated $(VLSI)$ technology $[12]$, it is feasible to implement *SOM* network on a chip. However, when using the *SOM* chip, some intrinsic faults may occur due to physical phenomena such as heat. In this paper, we study the problem of how to create fault immunization in *SOM* network on the computational level prior to its *VLSI* implementation. The fault immunization problem is possibly transformed to an optimization problem.

## 1.1 Problem Identification

The reliability of the network depends upon how the network is implemented or used. If the network is directly implemented on a *VLSI* circuit the reliability strongly depends on the design of the network. The problem of designing such a network is categorized into three approaches based on the solution and the constraints of the problems. These are

1. design of self-detection of faulty network by applying the techniques used in digital circuit design $[12]$,

2. design of a self-recovery network when there are faulty links and faulty neurons $[7]$, and

3. design of a fault tolerant network by injecting noise into input vectors, perturbing weight, and pruning some links during retraining [6].

Many fault models [2,3,4,5] are introduced and the solutions are proposed based on the assumption of the stuck-at-0 and stuck-at-1. Unlike those studies, we consider a general case of stuck-at-$a$ , where $a$ is a real value. There is another type of fault occurring at the weight vector. The fault is intermittent and prevented by a technique called *fault immunization*. Although these proposed techniques work well in most situations, there are still some possible improvements to enhance the robustness and reliability of the network.

The problem of fault tolerance and immunization has been widely studied as reported in [2,3,4,5,6,8,9,10,11]. Most studies concentrate on a feed-forward supervised network performing pattern classification. The only report closely related to the unsupervised network is in [7]. They proposed a fault tolerant technique of feed-forward networks called *weight shifting* to recover a self-organized network when some faulty links and/or neurons occur.

In this thesis, we focus on an unsupervised self-organization type network. A neuron with a self-organizing mechanism classifies its input data space into groups such that there are no common elements among groups. We consider the problem of enhancing the reliability of an *SOM* neural network by the technique of the synaptic links of each neuron, which is similar to the concept of biological immunization.

## 1.2 Objective

The main goals of this study are:

1. To develop an algorithm to enhance fault tolerance in a self-organizing artificial neural network.

2. To study the tolerance bound of a self-organizing artificial neural network under a given training set.

## 1.3 Scope of Work and Constraints

The following conditions are considered in this study:

1. No limitation is made on the number of synaptic links of any output neuron.
2. At any time, only one output neuron can be faulty.
3. No limitation is made on the input dimension and the value of each input element.
4. The value of each faulty synaptic link can be any real value, either positive or negative.
5. The synaptic weight adjustment is based on Kohonen self-organizing competitive mapping.

This thesis is organized as follows. Chapter II reviews the literature. Chapter III describes the theoretical background. In Chapter IV, a fault immunization model and the algorithm for the estimation of fault-tolerant capability are proposed. The experimental results of our evaluation are given in Chapter V. The discussion of the robustness of the fault tolerance *SOM* and finally the conclusions are given in Chapter VI.

# CHAPTER II

# LITERATURE REVIEW

## 2.1 Review of Literatures Related to Fault Tolerance and Immunization

R. K. Chun and L. P. McNamee [2] proposed the immunization of a neural network against hardware faults. A methodology and a set of computer-aided design tools for measuring and improving the fault tolerance characteristics of neural networks were presented. Two analysis programs have been developed using realistic fault models appropriate for emulating potential hardware failures. A modified training strategy, which reduced the network's sensitivity to faults, was also introduced. The process involved the injection of faults into the network during its training phase. The proposed scheme is said to be analogous to viral immunization in the biological domain because it is the neural network's own adaptive capability which is utilized to improve its fault tolerance characteristics. They proposed an original idea called *fault immunization*, which is based on trial-and-error retraining the network to obtain the best value of each weight vector.

C. Lursinsap and T. Tanprasert [9] applied the immunization concept in biological cells to enhance the fault tolerance capability in a perceptron-like neuron. In this paper, they considered only the case where each neuron separated its input vectors into two classes. A mathematical model was applied to get the cell immunization in terms of weight relocation and then a polynomial time

weight-relocating algorithm was proposed. This algorithm can be generalized for the network in which each neuron separates its input vectors into more than two classes. They extended Chun and McNamee's concept and proposed a mathematical model with the analysis of the fault immunization in terms of weight vector relocation for a perceptron unit.

K. Sunat and C. Lursinsap [10] extended Lursinsap and Tanprasert's concept mentioned above to capture the characteristics of the fault immunization. They investigated the capability of two random optimization techniques for the fault immunization improvement and proposed a new cost function, two hybrid training algorithms, which combined the target error function and the immunization function.

C. Lin and I. Wu [6] discussed the design of a fault tolerance network by injecting noise into input vectors, perturbing weight, and pruning some links during retraining. They examined a learning method that intended to maximize the fault tolerance. The method was based on the well-known back-propagation learning algorithm. During the training, each neuron was given a small probability to have a simulated failure. This modification ensures that the computation be distributed among different computing elements in the network and thus maximized the fault tolerance.

C. Neti, M. H. Schneider, and E. D. Young [3] presented a feed-forward neural network model with a guaranteed level of fault tolerance. The notion of fault tolerance and uniform fault tolerance in a neural network were defined and a method described to ensure that the estimated network exhibits fault tolerance. The problem of estimating weights was formulated as a large-scale nonlinear optimization problem. Numerical experiments indicated that the solutions with

uniform fault tolerance exist for the pattern recognition problem. Solutions derived by introducing fault tolerance constraints have better generalization properties than solution obtained via unconstrained back-propagation.

T. Tanprasert, C. Tanprasert, and C. Chidchanok [11] presented a technique for forcing the network to optimize its internal representation towards robustness and fault tolerance. The major idea was to force nonlinear classification on the linear classification problem. The technique introduced a concept of outpost vectors for hiding the unwanted linearly separable characteristics of the problem. The technique can be viewed as a variation of introducing noisy inputs. However, this technique was determined deterministically rather than randomly.

R. Singh, V. Cherkassky, and N. Perpanikolopoulos [14] proposed a method involving an iterative evolution of a piecewise-linear approximation of the shape skeleton by using a minimum spanning tree-based self-organizing mapping *SOM* . They considered the problem of computing the shape skeleton for shapes that lacked pixel level connectivity. Due to sparse shape, conventional skeletonization techniques performed poorly on such shapes.

Most studies are concerned with feed-forward supervised networks, and these are where most of this study's background information is from. Very little, however, has been written on unsupervised networks as below.

## 2.2 Review of Literatures Related to Unsupervised Network

C. Khunasaraphan, T. Tanprasert, and C. Lursinsap [7] proposed a fault tolerant technique for feed-forward neural networks called *weight shifting* and its analytical model. The technique was applied to recover a self-organized network when some faulty links and/or neurons occured during the operation. If some input links of a specific neuron were detected faulty, their weights would be shifted to the healthy links of the same neuron. On the other hand, if a faulty neuron was encountered, then the faulty neuron was treated as a special case of faulty links by considering all the output links of that neuron to be faulty. The aim of this technique is to recover the network in a short time without any retraining and hardware repair.

M. Yasunaga, I.Hachiya, K.Moki, and J.H. Kim [12] introduced a defect model of the SOM-WSI, and derive a critical-stuck-output of a defective neuron. They assumed that the output of defective neuron was stuck at a certain value (stuck output) and did not change through the *SOM* processing. In addition, they assumed that the defective neurons were concentrated in one place in the linear array, making a defective-neuron cluster.

# CHAPTER III

# THEORETICAL BACKGROUND

## 3.1 An overview of Neural Computing

Over the past four decades, the field of artificial intelligence (AI) has made a great progress toward automating human reasoning. Nevertheless, the tools of AI have been mostly restricted to sequential processing and only certain representations of knowledge and logic. A different approach to intelligent systems involves constructing computers with architectures and processing capabilities that mimic certain processing capabilities of the brain. The results are knowledge representations based on massive parallel processing, fast retrieval of large amounts of information, and the ability to recognize patterns based on experience. The technology that attempts to achieve these results is called neural computing or *artificial neural networks*, networks made up of neurons in much the same way as a brain. A biological neuron consists of four major components, namely, soma, axon, dendrite, and synapse. Figure 3.1 shows a portion of such a network. The input and output signals to the soma of a neuron are transmitted along the axon and dendrite. The synaptic resistance controls the strength of the signal. A neuron learns to generate a particular signal by adjusting the synaptic resistance.

In neural networks, we carry over this concept of synaptic resistance and refer to it as the weight of a neuron. There are three types of learning mechanisms dealing with the adjustment of a neuron weight. The first type is called *supervised learning*. A neuron is forced to generate a target signal associated with a specific input pattern and to reproduce this target signal whenever the specific input pattern occurs. The second type

of learning is called *unsupervised learning*. There is no target signal generated with a particular input pattern. A neuron competitively adjusts its weight value with the other neurons to make the value of its weight equal to the value of the input pattern. The last type of learning is called *reinforcement learning*. This type is a mixture of the first two types under the environment that the target is specified. In this research, we are concern with only the second type of learning.

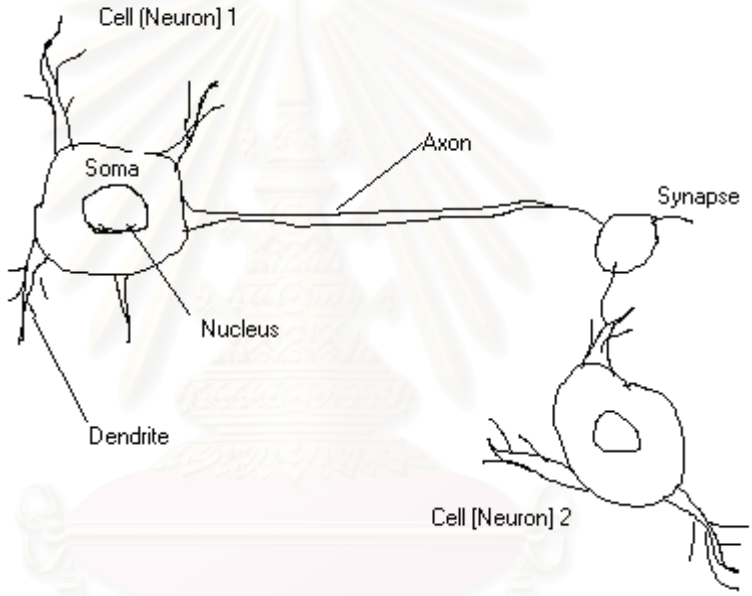



**Figure 3.1 Portion of a network: Two Interconnected Biological Cells.**

An artificial neural network is a mathematical model that emulates a biological neural network. The similarity between the biological neuron and the artificial neuron is summarized as follow.

| Biological | Artificial |
|---|---|
| Soma | Node |
| Dendrites | Input |
| Axon | Output |
| Synapse | Weight |

## 3.2 A Self-Organizing Mapping Artificial Neural Network

The learning pattern of an $SOM$ neural network is to arrange itself as a function of its inputs. This type of artificial neural network $(ANN)$ can be used for the quantification of a vector space. A number of input vectors are presented to the $ANN$, which has the characteristics of competitive learning as described by Kohonen [1] based on iterative computation of three steps:

1. The output neurons of the network compete among themselves to be activated, with the result that only one output neuron or one neuron per group is to be *on* at any one time.

2. The output neurons that *win* the competition are called *winner-take-all* neurons.

3. Updating weight by Kohonen learning rule.

In the most simple competitive learning networks, there is a single layer of output units $o_i$, each fully connected to a set of input $\xi_j$ via excitatory connections $w_{ij} \geq 0$, as shown in Figure 3.2.

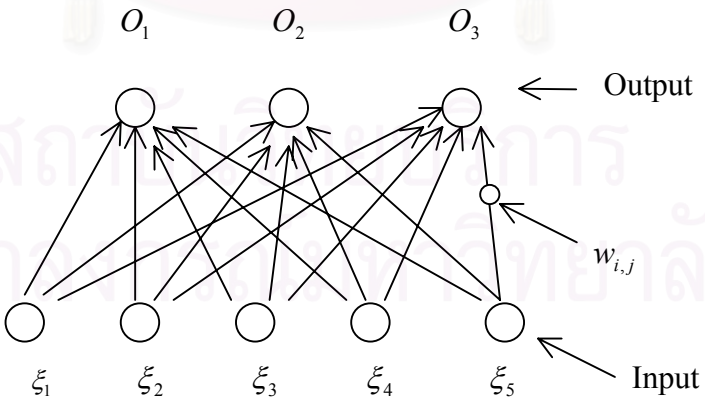



**Figure 3.2 A simple competitive learning network.**

Only one of the output units $i^*$, called the winner, can fire at a time. The winner is normally the unit with the largest net input

$$h_i = \sum_j w_{ij} \cdot \xi_j = w_i \cdot \xi \tag{3.1}$$

for the current input vector $\xi$. Thus, the condition of a winner $i^*$ is defined as follows

$$w_{i^*} \cdot \xi \geq w_i \cdot \xi \qquad \text{for all } i \tag{3.2}$$

with $O_{i^*} = 1$. If the weights for each unit are normalized so that $\| w_i \|$ for all $i$, then, (3.2) is equivalent to

$$\| w_{i^*} - \xi \| \leq \| w_i - \xi \| \quad \text{for all } i . \tag{3.3}$$

This says that the winner, $i^*$, is the unit with normalized weight vector $w_{i^*}$ closest to the input vector $\xi$. The winner-take-all network uses the criterion (3.2) or (3.3). The problem is how to find the clusters in the input data and adjust the weight vectors $w_i$ accordingly. We start with small random values for the weights. Then, a set of input patterns $\xi$ is applied to the network in succession. For each input, we find the winner $i^*$ among the outputs and then update the weight $w_{i^*}$ for the winning unit only to make the $w_{i*}$ vector closer to the current input vector $\xi$. Each weight element $w_{i^*j}$ of winning neuron $i^*$ is updated by adding to it an amount proportional to the difference between the input and the weight as calculated in equation (3.4)

$$\Delta w_{i^*j} = \eta \cdot \left( \xi_j - w_{i^*j} \right) \tag{3.4}$$

where $\eta$ is a learning rate.

## 3.3 An Example of Self-Organizing Mapping Neural Network

Given the following set of vectors in a 5-dimensional space, a simple competitive learning network is used to classify the vectors into three classes, namely, $w_1, w_2, w_3$.

$$\xi_1 = 10101 \quad \xi_2 = 11011 \quad \xi_3 = 01010$$
$$\xi_4 = 00011 \quad \xi_5 = 10010 \quad \xi_6 = 01100$$
$$\xi_7 = 01001 \quad \xi_8 = 11011 \quad \xi_9 = 10001$$

The network consists of one input layer and one output layer. An output neuron $i^*$ is the winner for the current input vector $\xi$ if it satisfies the following condition

$$w_{i^*} \cdot \xi \geq w_i \cdot \xi \text{ for all } i$$

where $w_{i^*}$ is the weight vector of the winner unit $i^*$. The classes are established by using the following steps.

1. Randomly choose initial weights. Each row is a weight vector of an output neuron.

$$w_1 = [0.7\ 0.3\ 0.7\ 0.3\ 0.7]^T,$$
$$w_2 = [0.3\ 0.8\ 0.7\ 0.6\ 0.3]^T,$$
$$w_3 = [0.3\ 0.3\ 0.3\ 0.7\ 0.8]^T$$

$W$ is the matrix formed from the transposed weight vectors.

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \end{bmatrix} = \begin{bmatrix} 0.7 & 0.3 & 0.7 & 0.3 & 0.7 \\ 0.3 & 0.8 & 0.7 & 0.6 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.7 & 0.8 \end{bmatrix}$$

2. Present vector $\xi_1 = 10101$ to the network.

First define a competitive transfer function, $compet\left(W \cdot \xi_1\right)$, which assigns an output of 1 to the neuron whose weight vector points in the direction closest to the input vector.

$$a = compet\left(W \cdot \xi_1\right)$$

$$a = compet\left\{\begin{bmatrix} 0.7 & 0.3 & 0.7 & 0.3 & 0.7 \\ 0.3 & 0.8 & 0.7 & 0.6 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.7 & 0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}\right\}$$

$$a = compet\left\{\begin{bmatrix} 2.1 \\ 1.3 \\ 1.4 \end{bmatrix}\right\} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The weight vector of the first neuron is closest to $\xi_1$. So, it wins the competition $\left(O_{1^*} = 1\right)$.

3. We now apply the Kohonen learning rule to the winning neuron with a learning rate of $\eta = 0.2$.

$$w_{1^*}^{new} = w_{1^*}^{old} + \eta \cdot \left(\xi_1 - w_{1^*}^{old}\right)$$

$$w_{1^*}^{new} = \begin{bmatrix} 0.7 \\ 0.3 \\ 0.7 \\ 0.3 \\ 0.7 \end{bmatrix} + 0.2\left\{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.7 \\ 0.3 \\ 0.7 \\ 0.3 \\ 0.7 \end{bmatrix}\right\} = \begin{bmatrix} 0.76 \\ 0.24 \\ 0.76 \\ 0.24 \\ 0.76 \end{bmatrix}$$

The Kohonen rule moves $w_{1^*}$ closer to $\xi_1$. If we continue choosing input vectors at random and presenting them to the network, then at each iteration the weight vector closest to the input vector will move toward that vector. Similarly, we find that the weight vectors $\xi_1$ and $\xi_9$ belong to $w_1$. The weight vectors $\xi_3, \xi_6$ and $\xi_7$ belong to $w_2$. The weight vectors $\xi_2$, $\xi_4$, $\xi_5$ and $\xi_8$ belong to $w_3$. Eventually, each weight vector will point at a different cluster of input vectors. Each weight vector becomes a prototype for a different cluster.

## 3.4 Simulated Annealing Search

Simulated Annealing$(SA)$ is a technique for solving combinatorial optimization problems such as minimizing functions of many variables. Typically, this involves finding a configuration of parameters $(x_1, x_2, x_3, ....., x_n)$ that minimizes an objective function.

The algorithm is a modified version of the iterative improvement algorithm, which involves starting with an existing non-optimal configuration and perturbing it in some small way. If this new configuration (solution) is better than the old one then accept it and start again as shown in Figure 3.3.

A new configuration is accepted if it is an improvement on the old one. However, this can cause problems, as shown in Figure 3.3. Suppose that we have an initial configuration $E_{start}$. We perturb $E_{start}$ and accept the better solutions by moving downhill only. We eventually arrive at point A and cannot go anywhere because uphill moves are not allowed. We get stuck in a local minimum even though the global minimum is at point B.

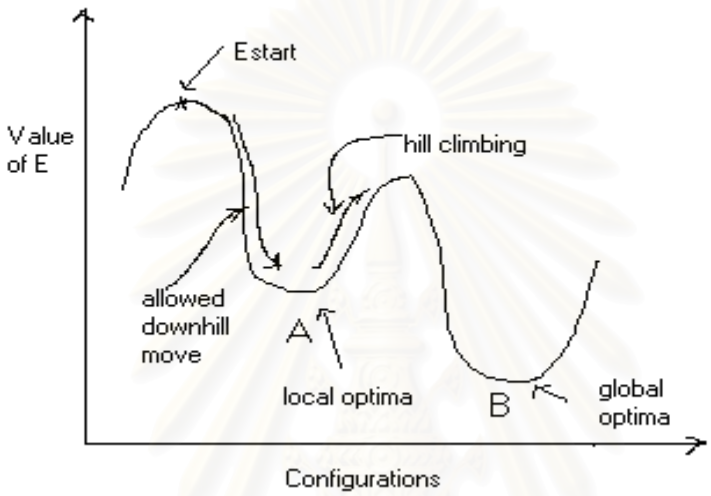



**Figure 3.3 Concept of Hill climbing to achieve the global optimal.**

We do not want to perform large perturbations because we would be jumping around in the solution space and "missing" the minimum. So we need some controlled way to avoid becoming stuck in a local minimum. This can be achieved by occasionally accepting uphill moves. This would allow us to jump out of A and downhill to B. This is achieved by accepting the configuration if $\Delta E < 0$ or accepting with probability $p = e^{\frac{-\Delta E}{T}}$ for some constant $T$.

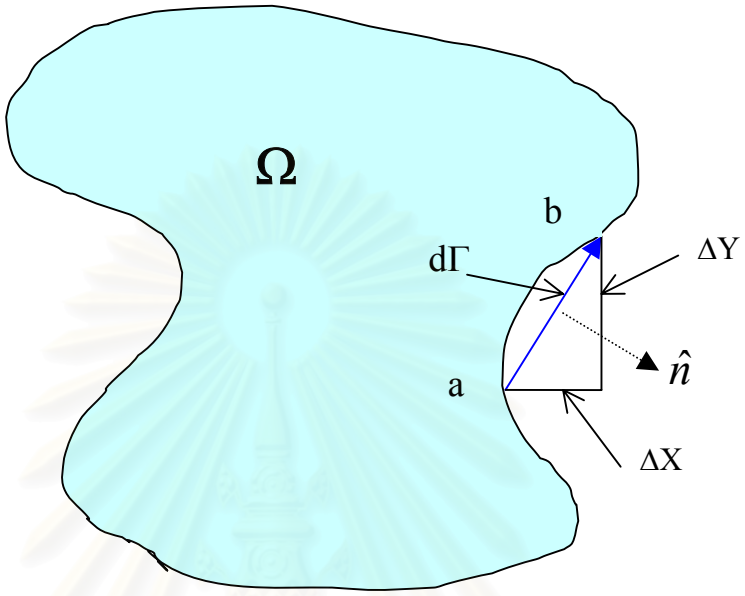
As the algorithm proceeds we hopefully progress nearer the optimal solution, i.e., $\Delta E$ decreases. However, since $T$ is fixed, $p = e^{\frac{-\Delta E}{T}}$ will increase (toward 1). As we get nearer the optimal value we are more likely to accept an uphill move and possibly miss the optimal solution. So we vary the temperature $T$ from a high to a low temperature.

## 3.5 Boundary Integrals

To maximize the area of the fault free space, we must first find an expression for this area. Since fault-free space is not a geometrical form, the area to be measured can be evaluated by the method of integration. For two-dimensional cases, the calculation of the fault-free space using boundary integrals is much faster than that using basic integrals. The concept of boundary integral is described below.

Domain integrals can be transformed into boundary integrals by using Green's theorem. By discretising the boundary into small segments, and each segment is approximated by an element, a straight line joining two end points of the segment, the transformed boundary integrals can be evaluated on each element.

The next steps showing how to derive boundary integrals for two-dimensional problem are illustrated.




$$\text{Area } \Omega = \iint_{\Omega} 1 \, d\Omega$$

**Figure 3.4 A domain $\Omega$ of which the area will be measured.**

Suppose there is a domain $\Omega$ of which the area will be measured.

Select two points: $a = (x_a, y_a)$ and $b = (x_b, y_b)$.

Connecting the two points $a$ and $b$, $\vec{ab} = (x_b - x_a)\vec{i} + (y_b - y_a)\vec{j} = \Delta x\vec{i} + \Delta y\vec{j}$ where $\vec{i} = (1,0)$, $\vec{j} = (0,1)$.

Taking infinitesimal limits of $\Delta x \cdot \vec{i} + \Delta y \cdot \vec{j}$, then $\vec{ab}$ becomes $d\vec{\Gamma} = dx\vec{i} + dy\vec{j}$ where $d\Gamma$ represents an infinitesimal segment of the boundary and the vector $d\vec{\Gamma}$ is tangential to $\Gamma$ at point $a$.

Therefore, $d\Gamma \equiv \left\| d\vec{\Gamma} \right\| = \sqrt{(dx)^2 + (dy)^2}$.

A unit vector in the tangential direction to boundary can be defined at a given boundary point as

$$\hat{\Gamma} = \frac{d\vec{\Gamma}}{d\Gamma} = \left(\frac{dx}{d\Gamma}\right)\vec{i} + \left(\frac{dy}{d\Gamma}\right)\vec{j}$$

Thus, a unit vector normal to the boundary and pointing outside the domain $\hat{n} = (n_x, n_y)$ may be defined in term of its directional cosines as follows;

$$\hat{\Gamma} \cdot \hat{n} = 0$$

$$\left(\frac{dx}{d\Gamma}, \frac{dy}{d\Gamma}\right) \cdot (n_x, n_y) = 0$$

$$\frac{dx}{d\Gamma} n_x + \frac{dy}{d\Gamma} n_y = 0$$

Choose $n_x = \frac{dy}{d\Gamma}$ and

$$n_y = -\frac{dx}{d\Gamma}$$

It is noted that $|\hat{n}| = 1$.

**Theorem** (Green 's theorem)

Let $f$ be a function from $\Omega \subseteq R^2$ (which is closed and has an enclosed boundary $\Gamma$) to $R$ and $\frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2}$ are continuous on $\Omega$,

then $\iint_{\Omega} \nabla^2 f \, d\Omega = \oint_{\Gamma} \frac{df}{dn} \, d\Gamma$ , $\oint_{\Gamma} d\Gamma$ is the line integral around $\Gamma$,

where $\frac{df}{dn} = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \cdot (n_x, n_y) = \frac{\partial f}{\partial x} n_x + \frac{\partial f}{\partial y} n_y$.

If $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \ldots \cup \Gamma_N$ where $\Gamma_i$ is a line segment joining points $\vec{x}_i = (x_i, y_i)$ and $\vec{x}_{i+1} = (x_{i+1}, y_{i+1})$,

then $\Gamma_i(\xi) = (x(\xi), y(\xi))$ where $\xi \in [0,1]$

and $x(\xi) = x_i(1-\xi) + x_{i+1}\xi$

$y(\xi) = y_i(1-\xi) + y_{i+1}\xi$.

If $n_x^i = \frac{dy_i}{d\Gamma_i} = \Delta y_i \frac{1}{J_i}$

$n_y^i = \frac{dx_i}{d\Gamma_i} = \Delta x_i \frac{1}{J_i}$

where $d\Gamma_i = \sqrt{(\Delta x_i)^2 + (\Delta y_i)^2} = J_i$

thus,

$$\int_{\Gamma_i} \frac{df}{dn^i} d\Gamma = \int_{\Gamma_i} \left( \frac{df}{dx} n_x^i + \frac{df}{dy} n_y^i \right) d\Gamma .$$

Since $d\Gamma = \left( \sqrt{\left(\frac{dx}{d\xi}\right)^2 + \left(\frac{dy}{d\xi}\right)^2} \right) d\xi = J_i \, d\xi$,

$$\int_{\Gamma_i} \frac{df}{dn^i} d\Gamma = \int_0^1 \left( \frac{\partial f}{\partial x} \Delta y_i \frac{1}{J_i} - \frac{\partial f}{\partial y} \Delta x_i \frac{1}{J_i} \right) J_i d\xi =$$

$$\int_0^1 \left( \frac{\partial f}{\partial x}(x(\xi), y(\xi)) \Delta y_i - \frac{\partial f}{\partial x}(x(\xi), y(\xi)) \Delta x_i \right) d\xi .$$

Therefore,

$$\iint_{\Omega} \nabla^2 f d\Omega = \sum_{i=1}^{N} \int_0^1 \left( \frac{\partial f}{\partial x}(x(\xi), y(\xi))\Delta y_i - \frac{\partial f}{\partial y}(x(\xi), y(\xi))\Delta x_i \right) d\xi .$$

As we are looking for the area of $\Omega$, and we have

$$\text{Area } \Omega = \iint_{\Omega} 1 \, d\Omega,$$

We can set the function $f(x, y)$ to be equal to $\frac{x^2}{2}$. Then we have

$\nabla^2 f(x, y) = 1$, and

$$\frac{\partial f}{\partial x}(x, y) = x \ , \ \frac{\partial f}{\partial y}(x, y) = 0 .$$

As a result,

$$\frac{\partial f}{\partial x}(x(\xi), y(\xi)) = x_i(1-\xi) + x_{i+1}\xi = x_i + (x_{i+1} - x_i)\xi$$

Consequently,

$$\iint_{\Omega} \nabla^2 f d\Omega = \sum_{i=1}^{N} \int_0^1 \left( (x_i + (x_{i+1} - x_i)\xi)\Delta y_i \right) d\xi$$

$$\iint_{\Omega} 1 \, d\Omega = \sum_{i=1}^{N} (y_{i+1} - y_i)\left( x_i + \frac{1}{2}\Delta x_i \right)$$

$$\text{Area } \Omega = \frac{1}{2}\sum_{i=1}^{N} (y_{i+1} - y_i)(x_i + x_{i+1})$$

The final equation obtained here will be used to measure the area and volume of fault-free space for two and three-dimensional vectors.

## 3.6 Multidimensional Integrals

Here, the measurement of fault-free space is generalized to n-dimensional space. Therefore, multidimensional integrals are applied here. International standard four-dimensional input vectors were used as well as the five dimensional input vectors of a color image. The concept is expressed as below.

Numerical integration, which is also called quadrature, is based on adding up the value of the integrand at a sequence of abscissas within the range of integration. As an example, we will consider the case of a four-dimensional integral in $x, y, z, v$ space. Two, three dimensional, or more than four-dimensional integration, are entirely analogous.

The first step is to specify the region of integration by

1. Its lower and upper limits in $x$, which we will denote $x_1$ and $x_2$,

2. Its lower and upper limits in $y$ at a specified value of $x$, denoted $y_1(x)$ and $y_2(x)$,

3. Its lower and upper limits in $z$ at specified $x$ and $y$, denoted $z_1(x, y)$ and $z_2(x, y)$,

4. Its lower and upper limits in $v$ at specified $x$, $y$ and $z$, denoted $v_1(x, y, z)$ and $v_2(x, y, z)$.

In other words, find the numbers $x_1$ and $x_2$, the functions $y_1(x)$, $y_2(x)$, $z_1(x, y)$, $z_2(x, y)$, $v_1(x, y, z)$ and $v_2(x, y, z)$ so that

$$I = \iiint\int dx\, dy\, dz\, dv\, f(x, y, z, v)$$

$$= \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dz \int_{v_1(x,y,z)}^{v_2(x,y,z)} dv\, f(x, y, z, v)$$

Now we can define a function $F(x, y, z)$ that does the innermost integral,

$$F(x, y, z) \equiv \int_{v_1(x,y,z)}^{v_2(x,y,z)} f(x, y, z, v)\, dv$$

and a function $G(x, y)$ that does the integral of $F(x, y, z)$,

$$G(x, y) \equiv \int_{z_1(x,y)}^{z_2(x,y)} F(x, y, z)\, dz$$

and a function $H(x)$ that does the integral of $G(x, y)$,

$$H(x) \equiv \int_{y_1(x)}^{y_2(x)} G(x, y)\, dy$$

and finally our answer as an integral over $H(x)$

$$I = \int_{x_1}^{x_2} H(x)\, dx$$

# CHAPTER IV

# SOM FAULT IMMUNIZATION MODEL AND ALGORITHM

## 4.1 Mathematical Model of SOM Fault Immunization

As we have seen in Chapter III, the mathematical model of SOM fault immunization is based on the unsupervised learning of an SOM network. Before going into any further discussion, we should briefly summarize the learning process of an SOM network. In the unsupervised learning of an SOM network, there is a single layer of output units, each fully connected to a set of input via excitatory connections. Only one of the output units, called the winner, can fire at a time. Let $W_i = (w_{i,1}, w_{i,2}, \ldots\ldots, w_{i,m})$ be the weight vector of neuron $i$, $W_{j^*}$ the weight vector of winning neuron $j$, and $I$ the input vector set. The winner is normally the unit with the largest net input.

$$h_i = \sum_k W_i \cdot \xi_k \qquad \text{for the current input vector } \xi_k \ .$$

Thus,

$$W_{j^*} \cdot \xi_k \geq W_i \cdot \xi_k \qquad \text{for all } i \, .$$

If the weights for each unit are normalized so that $\| W_i \| = 1$ for all $i$, then, the above relation is equivalent to

$$\| W_{j^*} - \xi_k \| \leq \| W_i - \xi_k \| \qquad \text{for all } i \, .$$

This says that the winner is the unit with normalized weight vector $W_{j^*}$ closest to the input vector $\xi_k$. Therefore, each weight vector $W_i$ is a class representative. The network stops learning if each weight vector $W_i$ is a winner with respect to input vectors in its class and the intersection of all classes is an empty set class. Suppose that weight vector $W_i$ is the representative of a subset of input vectors, $I_i$, when the network is implemented on a VLSI chip, the value of each weight vector can be gradually altered by various causes as heat, electromagnetic field, or radiation. The problem is how to immunize the chip to make it tolerant to the fault caused by the disturbance of the weight vector. The relevant definitions are given as follows.

**Definition 4.1** Let $I_i$ be a subset of input vectors whose representative is $W_i$ and $I_i'$ be a subset of input vectors whose representative is $W_i + \varepsilon_i$. The vector $\varepsilon_i$ is in between the lower bound $-L_i$ and upper bound $U_i$, $-L_i \le \varepsilon_i \le U_i$. Weight vector $W_i$ has fault immunization if $I_i = I_i'$. $W_i$ is called a fault immunized weight vector.

**Definition 4.2** A fault free space, $F(W_i)$, with respect to the fault immunized weight vector $W_i$ is the cardinality of a set of all weight vectors obtained from $W_i + \varepsilon_i$ for all possible values of $\varepsilon_i$ and the members in the class of $W_i$ and the members in the class of $W_i + \varepsilon_i$ are the same.

Based on the mentioned definitions, we formulate the problem of fault immunization of SOM network as an optimization problem as follows.

**Problem formulation:** Given a set of input vector and the number of classes, find $W_i$, for all $i$, such that

1. $|F(W_i)|$ is maximum.
2. Each element of $L_i$ is maximum.
3. Each member of $U_i$ is maximum.

Consider an example of fault free space and fault immunization as shown in Figure 4.1 and 4.2. There are five classes in this case. In Figure 4.1, the shapes shown are the fault free spaces when the weight vector of each class is not at the best location to achieve the maximum fault free space. The best location of each weight vector is illustrated in Figure 4.2. It can be noted that the fault free space of each class is apparently increased and the members of each class remain the same.

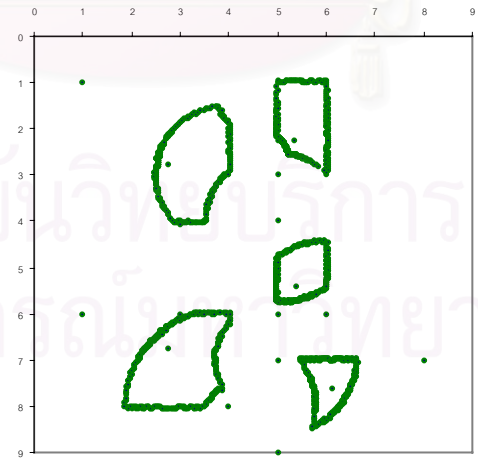



**Figure 4.1 An example of the fault free space when $W_i$ is not at the location of maximum $|F(W_i)|$.**


**Figure 4.2 New weight location of the example in Figure 4.1 when maximum $|F(W_i)|$ is achieved.**

## 4.2 Error Function and Random search

In order to achieve the maximum fault immunization of any SOM network, there are two problems to be considered. The first problem is the unsupervised classification of input vectors and maintaining the correct classes as all corresponding weight vectors are relocated to the locations of maximum fault free space. The second one concerns how to relocate all weight vectors to achieve maximum fault free spaces for all weight vectors.

Let $W_j$ be the weight vectors of $I_j(t)$, a subset of input vectors at time $t$. $I_j(t)$ will be classify according to $W_j$, into $C$ classes. The two problems are combined and solved with the following error functions.

$$E(t) = a\,E_{class}(t) + (1-a)\,E_{fault}(t) \tag{4.1}$$

$$E_{class}(t) = \sum_{j=1}^{C} \left| I_j(t) \cap I_j(t-1) \right| \tag{4.2}$$

$$E_{fault}(t) = \sum_{j=1}^{C} \frac{1}{\left|F(W_j)\right|} \tag{4.3}$$

where $E_{class}$ is the class error, $E_{fault}$ is the fault free space error, and $a$ is a constant such that $a \in [0,1]$. The value of each weight vector is adjusted to minimize $E(t)$. The adjustment is performed until $E(t) \le k$ or $t \le T$, where $k$ and $T$ are constants. Fault immunization consists of the steps to minimize $E_{class}$ followed by the steps to find the new location of each weight vector such that $E_{class} = 0$ and $E_{fault} < k$. The value of $k$ may be zero. The condition $E_{class} = 0$ implies that each class reaches its stable state. Set $I_i$ will be used to find an optimal location of $W_i$. Since the shape of each $F_i$ is unknown, a random search must be employed to find the new location of $W_i$ and boundary of $F(W_i)$. Here, the technique of simulated annealing is applied for this purpose. The detail of the fault immunization is given as follows.

### 4.2.1 SOM Fault Immunization Algorithm

The algorithm, first, finds the location of each $W_i$ to achieve $E_{class} = 0$. Then the algorithm applies simulated annealing search to relocation each $W_i$ to achieve minimum $E_{fault}$. The details are as follows.

1. Initialize all weight vectors to small values and $a$ =1.
2. Set iteration number as $m$ = 1.
3. Present the vector $\xi_k$ to the network.
4. **For** each input pattern,
5. Compute the Euclidean distance $(D)$ between each input vector $(\xi_k)$ and weight vector $(W_i)$

$$D = \| \xi_k - W_i \|$$

6. Compare the winning neuron with weight vector $w^*$, which has the closest weight to the input vector, is chosen as:

$$\| \xi_k - W_{j^*} \| = \min_i \| \xi_k - W_i \|$$

7. Apply the Kohonen learning rule to the winning neuron with leaning rate, $\eta$ <<1. The winner $W_{j^*}$ updates their weights as:

$$W_{j^*}^{new} = W_{j^*}^{old} + \eta \cdot \left( \xi_k - W_{j^*} \right)$$

**End for** (each input pattern)

8. Set the new iteration number as $m = m + 1$. Iteratively calculate of the steps 5, 6 and 7 until $E_{class} = 0$.
9. Let $I_{j^*}$ be the subset of input vectors designated by the winner weight vector $W_{j^*}$.

10. Use simulated annealing search to generate a new location of each weight vector $W_{j^*}$ such that $I_{j^*}$ remains the same.

11. Compute $E_{fault}(t) = \sum_{j=1}^{C} \frac{1}{|F(W_j)|}$.

12. **If** $E_{fault} > k$ **then** repeat step 10 to 12.

### 4.2.2 Optimal Weight Relocation Algorithm by Simulated Annealing Search

Let $P$ be a random number where $0 \le p \le 1$, $\Delta_i$ be the change of fault free space for class $i$ and $\alpha$ be a constant where $0 \le \alpha \le 1$.

1. Set $T$ to a fixed value or a temperature and $E_{fault}^{old}$ to a high value.
2. **While** $T > T_{\min}$ **do**
3. **For** all classes $i = 1$ **to** $C$
4. Generate a new location $W_i^{'}$.
5. Compute a new error $E_{fault}^{new}$ using equation $(4.3)$.
6. Compute the change of the error, $\Delta E_{fault} = E_{fault}^{new} - E_{fault}^{old}$.
7. **If** $\Delta E_{fault} < 0$ or $P \le e^{-\Delta E_{fault}/T}$ **then**
8. Set $W_i = W_i^{'}$
9. Set $E_{fault}^{old} = E_{fault}^{new}$

**End if**

**End for**

10. Set $T = \alpha T$

**End while**

### 4.2.3 Algorithm for finding Fault Free Space $|F(W_i)|$

Let $W_i = (w_{i,1}, w_{i,2}, ....., w_{i,m})$ be the weight vector of class $i$, $I_i$ be the subset of input vectors in class $i$, $L_j$ be the lower bound of dimension $j$, and $U_j$ be the upper bound of dimension $j$. The fault free space with respect to $W_i$ can be found by gradually moving $W_i$ to a new location and verify whether $I_i$ changes its members. The location that $I_i$ remains unchanged is called a *valid location*. The value of $w_{i,j}$ must be constrained by the value of $L_j$ and $U_j$ in all dimensions. All the valid locations will be $|F(W_i)|$.

### Fault-free space algorithm for two-dimensional vectors

This algorithm finds the boundary of fault-free space for two-dimensional vectors and then applies boundary integrals to compute fault-free area.

1. Start with a weight vector, $W_i$, a step size length $\delta$, and an azimuth angle, *azimuth*.
2. Set an initial step size length, $r$, and an initial azimuth angle as $\alpha = 0$.
3. Set $N$ = total of elements; $N = 360^{\circ} / azimuth$.
4. **For** $l = 1$ **to** $N$
5. **Repeat**
6. Find a new location by setting $X_l = r \cdot \cos(\alpha) + W_i$ and $Y_l = r \cdot \sin(\alpha) + W_i$.

7. Checks **if** there is no misclassification and $w_j \in [L_j, U_j]$ **then** set $r = r + \delta$.

**Until** any constraint is violated.

8. Store the location obtained from the above step and initialize step size length, $r$. And set $\alpha = \alpha + azimuth$.

**End for**

9. Compute fault-free area $(FA)$ as:

$$FA_{class\ i} = \frac{1}{2}\sum_{l=1}^{N}(y_{l+1} - y_l)\cdot(x_l + x_{l+1})$$

This algorithm can be extended into three dimensions by adding an angle of elevation and summing over it.

## Multidimensional Integrals Algorithm

This algorithm finds the boundary of fault-free space for n-dimensional vectors, then calculates fault-free space.

1. Set step length $\delta_j = (\delta_1, \delta_2, \ldots, \delta_n)$.
2. Set an initial number of weight vectors that satisfied the constraint, $ncount_i$.
3. Set each weight element $w_{i,j} = L_j + \frac{\delta_j}{2}$.
4. **For** all dimensions $1 \le j \le n$
5. Fixed the value of each $w_{i,k}, 1 \le k \le n$ and $k \ne j$ .

6. Set $w_{i,j} = w_{i,j} + \delta_j$ .

7. Checks **if** there is no misclassification and $w_i \in \lfloor L_j, U_j \rfloor$ **then**

   Set $ncount_i = ncount_i + 1$

   **End for**

8. Compute fault-free space $(FS)$ as:

$$FS_{class\ i} = ncount_i \cdot \delta_i$$

# CHAPTER V

# EXPERIMENTAL RESULTS

The algorithm is tested with four examples. The data in the first example are in a two dimensional space. There are five classes. Figure 4.1 shows the fault free space when each weight vector is at a non-optimal location and Figure 4.2 shows the fault free area when the weight vectors are at optimal locations. Table 5.1 indicates value of each weight vector at the location obtained from *SOM* learning, called first weight, and the best location, called optimal weight, and the fault free spaces in both situations. The following abbreviations are used in the Table:

| | |
|---|---|
| 1st Wt | first weight |
| 1st FS | fault free space of the first weight |
| Opt Wt | Optimal weight |
| Opt FS | fault free space of the optimal weight |

**Table 5.1 Locations of the first weight vectors and optimal weight vectors as well as their fault free spaces for five classes.**

| Class | 1st Wt | | Opt Wt | | 1st FS | Opt FS |
|---|---|---|---|---|---|---|
| 1 | 2.734 | 2.771 | 2.874 | 3.189 | 2.803E+00 | 4.905E+00 |
| 2 | 5.332 | 2.262 | 5.814 | 1.582 | 1.688E+00 | 1.732E+00 |
| 3 | 5.369 | 5.402 | 5.822 | 5.061 | 1.148E+00 | 1.255E+00 |
| 4 | 2.771 | 6.738 | 2.546 | 7.628 | 3.108E+00 | 3.108E+00 |
| 5 | 6.107 | 7.598 | 6.182 | 7.573 | 9.838E-01 | 2.312E+00 |

The total fault free space before the immunization is 9.730483E+00 and after the immunization is 1.331144E+01.

The second example is the classical Iris data [13] in a four dimensional space. This consists of details of petal and sepal dimensions of three different species of Iris flower, namely, Iris sestosa, Iris versicolor and Iris virginica. They are classified into three classes. Table 5.2 shows the value of each weight vector at the location obtained from *SOM* learning, called first weight, and the best location, called optimal weight, and the fault free spaces in both situations.

**Table 5.2 Locations of the first weight vectors and optimal weight vectors as well as their fault free spaces for classical Iris data.**

| Class | 1st Wt | | | | Opt Wt | | | | 1st FS | Opt FS |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5.006 | 3.450 | 1.452 | 0.244 | 5.042 | 3.372 | 1.468 | 0.247 | 2.048E-02 | 2.048E-02 |
| 2 | 5.991 | 2.600 | 4.001 | 1.330 | 5.996 | 2.600 | 3.975 | 1.337 | 4.096E-05 | 1.228E-04 |
| 3 | 6.698 | 2.999 | 5.555 | 2.199 | 6.670 | 2.940 | 5.618 | 2.241 | 4.096E-05 | 8.192E-05 |

The total fault free space before the immunization is 2.056192E-02 and after the immunization is 2.068565E-02.

**Figure 5.1 Bird image.**

**Figure 5.2 Bird image with 100 classes.**

The third example is the image of a bird shown in Figure 5.1. Each pixel $p_i$ of the image is considered as a vector, $p_i = (x_i, y_i, g_i)$, in a three dimensional space. The location of $p_i$ is represented by coordinates $x_i$ and $y_i$. The gray-scaled color of the pixel is the value of $g_i$. The pixel is classified into 100 classes. The total fault-free space before immunization is 1008.876883 and after immunization is 1274.772590.

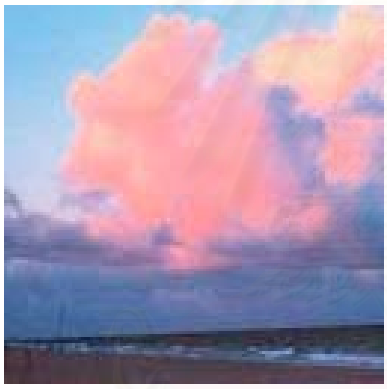


**Figure 5.3 Beach cloud image.**

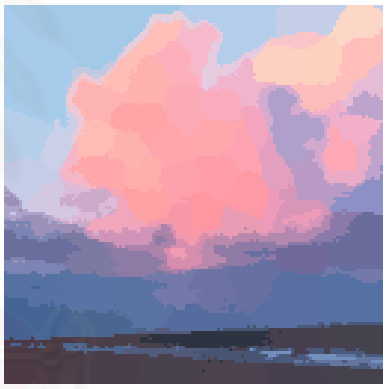


**Figure 5.4 Beach cloud image with 100 classes.**

The last example is the image of beach cloud shown in Figure 5.3. Each pixel $p_i$ of the image is considered as a vector, $p_i = (x_i, y_i, r_i, q_i, b_i)$, in a five dimensional space. The location of $p_i$ is represented by coordinates $x_i$ and $y_i$. The 256 level color of the pixel is the value of $r_i, q_i, b_i$. The pixel is classified into 100 classes. The total fault-free space before immunization is 4.483630 and after immunization is 4.548831. From the experiments, the

immunization improves the fault free space but the percentage of improvement depending upon the density of the input vectors. In the case of Iris data and beach cloud image, the density of the data is high while the density of the bird image is low.

For the last two examples, we applied the fault immunization to enhance the reliability of the compressed image. The compression is of a lossy type, that is, some information is lost during compression, and is based on the vector quantization concept. The first image is a bird image and the second image is a cloud image. Without loss of the general fault immunization concept, we limit our experiment to 100 classes due to the computational time.

# CHAPTER VI

# CONCLUSION

The problem of fault immunization in an *SOM* neural network and an immunization technique are introduced. The efficiency is measured in terms of the amount fault free spaces, which depends upon the density of the input vectors. One direct application of these studies is to enhancing the reliability of the lossy image compression by a technique of vector quantization. Each weight vector acts as a center of the cluster of the image pixels. If there are some faults occurring in the weight vectors these vectors still correctly represent the same clusters.

The problems studied as well as the results proposed in this thesis are still in the infancy state. Yet, there are other interesting further studies, that are listed as follows:

1. In practical application, the situation of multiple faulty neurons should be considered.
2. Since Simulated Annealing is a statistical approach, an important approach would be to develop techniques for control parameter $T$ used in the algorithm.
3. Other optimization techniques besides Simulated Annealing such as Genetic Algorithm can also be used.
4. For conventional neural networks it takes up a lot of the computational time so that the parallel computing should be developed.

# REFERENCES


[1] T. Kohonen. "The Self-Organizing Map". Proceedings of the IEEE, Vol. 78, No. 9, September 1990: pp. 1464-1480.

[2] R. K. Chun and L.P. McNamee. "Immunization of Neural Networks Against Hardware Faults". IEEE International Symposium on Circuits and Systems, Vol. 1, 1990: pp. 714-718.

[3] C. Neti, M.H. Schneider, and E.D. Young. "Maximally Fault Tolerant Neural Networks". IEEE Transactions on Neural Networks, Vol. 3, No. 1, 1992: pp. 14-32.

[4] P. Ruzicka. "Learning Neural Networks with Respect to Tolerance to Weight Error". IEEE Transactions on Circuits and Systems, Vol. 40, No. 5, May 1993: pp. 331-342.

[5] H. Elsimary, S. Mashall, A. Darwish, and S. Shaheen. "Performance Evaluation of Novel Fault Tolerance Training Algorithm". IEEE International Conference on Neural Networks, Vol. 2, 1994: pp. 856-861.

[6] C. Lin and I. Wu. "Maximizing Fault Tolerance in Multilayer Neural Networks". IEEE International Conference on Neural Networks, Vol. 1, 1994: pp. 419-424.

[7] C. Khunasaraphan, T. Tanprasert, and C. Lursinsap. "Recovering Faulty Self-Organizing Neural Networks by Weight Shifting Technique". IEEE World Congress on Computational Intelligence, Vol. 3, June 1994: pp. 1513-1518.

[8] W. Fornaciari and F. Salice. "A new Architecture for the Automatic of Custom Digital Neural Networks". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, December 1995: pp. 502-506.

[9] C. Lursinsap and T. Tanprasert. "Fault Immunization Technique for Artificial Neural Networks". IEEE International Conference on Neural Networks, Vol. 1, June 1997: pp. 302-307.

[10] K.Sunat and C. Lursinsap. "Learning Algorithm for Global Fault Immunization of Supervised ANN". IEEE Asia-Pacific Conference on Circuits and Systems, 1998: pp. 655-658.

[11] T.Tanprasert, C. Tanprasert, and C. Lursinsap. "Contour Preserving Classification for Maximal Reliability". IEEE International Joint Conference on Neural Networks, Vol. 2, 1998: pp. 1125-1130.

[12] M.Yasunaga, I.Hachiya, K.Moki, and J.H. Kim. "Fault-Tolerant Self-Organizing Map Implemented by Wafer-Scale Integration". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 6, No. 2, June 1998: pp. 257-265.

[13] James C. Bezdek, James M. Keller, Raghu Krishnapuram, Ludmila I. Kuncheva, and Nikhil R. Pal. "Will the Real Iris Data Please Stand up?". IEEE Transactions on Fuzzy Systems, Vol. 7, No. 3, June 1999: pp. 368-369.

[14] R. Singh, V. Cherkassky, and N. Perpanikolopoulos. "Self-Organizing Map for the Skeletonization of Sparse Shapes". IEEE Transactions on Neural Networks, Vol. 11, No. 1, January 2000: pp. 241-248.

# CURRICULUM VITAE

Rachaneewan Talumassawatdi was born in February 12, 1956. She received a bachelor degree in Science (Chemistry) from Faculty of Science, Ramkamheang University in 1979. Presently, she is affiliated with Bureau of the Royal Rainmaking and Agricultural Aviation, Ministry of Agricultural and Cooperatives on the position of research scientist (level 7). Her main responsibility is the analysis of radar data and satellite data for artificial rain prediction.