Chapter I
Introduction

1.1 Introduction to Fault Tolerance of Neural Networks

The fault tolerance of a neural network refers to the capability of the network to continue its proper operation while there exist internal failures or other anomalies in the hardware system on which it is implemented [1]. As the power of neural systems becomes more evident by the increasing of their sophisticated applications, their usage will become widespread. However, before neural networks can be approved to be useful in important or life-critical applications, it is necessary to study their performance under conditions that are less than the ideal conditions. Several causes of faults of a neural network have been studied by many researchers [2, 3, 4]. Some of these causes are:

1. The property of semiconductor matter, which is used for implementing the neural network circuits.

2. Operating conditions such as the temperature level, the supplied voltage, and the age of the device used for implementing the network.

Feltham and Maly [5] have shown that the weight vector is the most sensitive part to fault or damage. There are many fault models that have been proposed in the past e.g. stuck-at-0, stuck-at-1, stuck-at-x [6, 7, 8, 9]. However, the most popular one is the techniques of injecting the input noise and retraining the network. Another solution is based on the redundancy of the neurons [10]. These approaches are efficient and simple to implement.

There is another type of fault occurring at the weights [1, 11]. The fault is intermittent and can be prevented by a technique called Fault Immunization. Chun and McNamee [1] originally proposed the idea called fault immunization which is based on the trial-and-error of retraining the networks to obtain the best location of each weigh vector. In view of Chun and McNamee, the trained weights and the thresholds in the network can be viewed as "links" through which information can flow. Since these links define the performance and the functionality of the neural network, any perturbation to them, caused by a hardware malfunction, would jeopardize the proper operation of the network. Viewed in a generic manner, one possible method to model faults of hardware implementations of a neural network could be the deviation in weight and in threshold values after the network has been trained. Such a fault model would be simple to implement in a typical neural network simulation environment, and could be considered as a high-level and generalized case of many failure modes possible in digital, analog, and optical versions of hardware platforms. The link perturbation model could also measure the effects of incomplete training.

Chun and McNamee [1] proposed *"headroom"*, a measure for fault tolerance capability. Headroom addresses the question of how much each weight and threshold can be perturbed (in terms of percentage change) and still have the network producing the answers which are correct to some extent within a certain tolerance range. An acceptable extended output range, or noise margin of ± 0.2 is chosen for their design scenario. They found that error backpropagation, EBP, did not use fault tolerance as a criteria to be optimized and proposed a technique to improve fault tolerance capability. To desensitize the weights and thresholds against the possibility by noise, it is hypothesized that they should be exposed to noise during the training process. They proposed the concept of injecting simulated hardware faults into the network while being trained. This concept has an interesting biologically analogy–that of viral immunization.

The neural network is being "immunized" against faults by inoculating it with minor occurrences of faults during the training process. The network trained by adding a random amount of noise to each of its weights, one-at-a-time, in a cyclic fashion. The effect of noise is simulated using random perturbations induced by scaling the weights in the network, one at a time, either up or down by a certain maximum percentage on each pass of EBP training algorithm. The physical location of the noise is simulated to be roving in nature, moving from one weight to the next. In this manner, it is hoped that the effects of noise will be distributed equally throughout the network and will have the minimum impact upon normal convergence. Since a random number generator is used, zero-mean white noise can be approximated if a sufficiently large number of samples is taken, but it took very long time to converge and lack mathematical support.

Chun and McNamee did not provide any mathematical analysis model. Lursinsap and Tanprasert [11] extended their concept and proposed a mathematical model with analysis of the fault immunization in term of weight vector relocation for the perceptron unit, but their approach can be used only with one neuron (perceptron) unit and deteriorated the classification capability of neuron.

1.2 Contributions

The contributions of this thesis are as follows:

1. A feasible mathematical model for measuring fault immunization of supervised feedforward neural networks.

2. A feasible algorithm for enhancing fault immunization of supervised feedforward neural networks.

1.3 Scope of Study

This thesis extends the mathematical model of [11] for each neuron in a feedforward neural network whose input data are in the interval [0,1]. Here, the network is considered as data classifier in a real value space. The training set is expected to be fully representative of the inputs to the net during application. The results will be shown by the theorem-proving style, and the simulation results will be shown to confirm the theorems. Before we continue our description of these concepts, the feedforward neural networks and some learning algorithms will be summarized as follows:

1.3.1 The Feedforward Neural Networks

An artificial neural network, a mathematical computational model, consists of interconnected neurons. A neuron is modeled after McCulloch and Pitts' proposed model of neural activities in the human brain [12]. The model takes a simple form:

$$output \;=\; f(\sum_{i=0}^{n} w_i x_i) \tag{1.1}$$

where x_0 is the unity, w_0 is the threshold or bias, and f is called the *activation function*, which usually takes the form:

$$f(x) = A\, tanh(\frac{x}{T}) + B \tag{1.2}$$

where $A, T > 0$ and B is arbitrary. The most widely used activation functions are the ones with $(A,T,B)=(1,1,0)$ or $(\frac{1}{2}, 2, \frac{1}{2})$. These two variations of f are called the *tan-hyperbolic*, eq. 1.3, and the *logistic*, eq. 1.4, activation functions, respectively.

$$f(x) \;=\; tanh(x) \tag{1.3}$$

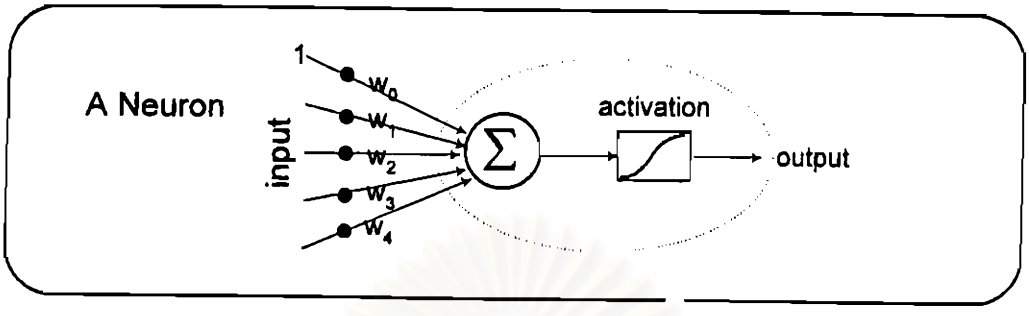

Figure 1.1: McCulloch and Pitts' Mathematical Model of a Neuron

$$f(x) = \frac{1}{1+e^{-x}} = \frac{1}{2}tanh\left(\frac{x}{2}\right) + \frac{1}{2} \tag{1.4}$$

A feedforward neural network is a form of interconnected neurons without having a feedback loop. A typical neural network [13] is shown in Figure 1.2. This type of neural network was shown by Cybenko [14] that it is a universal function approximator for any arbitrary real functions. Therefore we can view this type of network as an implementation of a real mapping g from its input to its output.

1.3.2 Concised Backpropagation Algorithm

One of the most widely used training algorithms for a feedforward neural network is the backpropagation algorithm proposed by Rumelhart [15]. This learning algorithm inspired an extensive attention in the neural network theory and its applications.

Suppose T is a set of training patterns, which is denoted as follows

$$T = \{p_1, p_2, \cdots, p_N\} \tag{1.5}$$

Each pattern p_i is a 2-tuples vector consisting of an input training vector x and a target vector t. The adjustable optimization parameters for the performance of a given neural network using a given training set T are the weights w_i (eq. 1.1). This learning scheme is called a supervised learning algorithm.

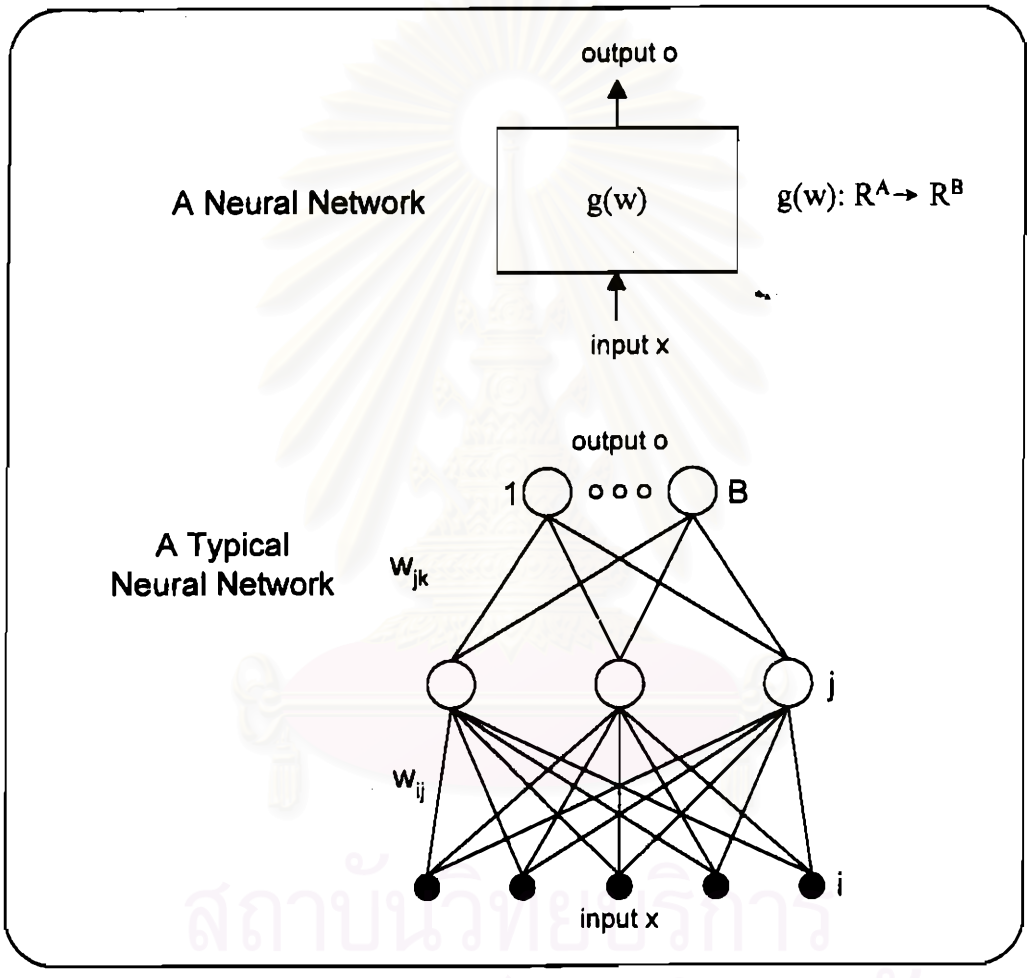

Figure 1.2: A Typical Neural Network

Rumelhart et al. [16] shows that the weight w should be adjusted to minimize the error function E, which is defined over all possible weight w and consists of two terms:

$$E(w) = E_t(w) + priors(w) = E_t(w) + P(w). \tag{1.6}$$

E_t is the data-fitting error function or target error function with variable vector w. The vector w is adjusted according to the difference between the actual output vector produced by the neural network and the corresponding target vector to optimized E_t.

The priors P, the network complexity penalty function, regulates the learning algorithm to find another possible characteristics. Typically, the data-fitting error is defined by employing the L^2 norm of the difference between the output vector o of the neural network and the target vector t associated with the input vector x. So the instantaneous data-fitting error function $\hat{E}_t$ for a particular pattern p_i, and the weight vector w can be defined as:

$$\hat{E}_t(w, p_i) = \frac{1}{2}||t_i - o_i||^2 \tag{1.7}$$

If the training samples in T (eq. 1.5) are drawn independently, then the appropriate data-fitting error function E_t over all the patterns is simply [16]:

$$E_t(w) = \sum_{i=1}^{N} \hat{E}_t(w, p_i) \tag{1.8}$$

Rumelhart et al. [16] uses a simple priors P as the "weight-decay factor".

$$P(w) = \frac{1}{\sigma^2} \sum_{i=1}^{M} w_i^2 \tag{1.9}$$

where $\sigma > 0$ and M=the number of weights. By using this priors function, the neural network prefers small weights around the origin.

There are two of the weight updating schemes [17, 13, 18]. On-line updating is based on the gradient of pattern-by-pattern scheme while batch updating is

based on the accumulating gradient of all patterns in the training set T to form a global gradient which is updated once at the end of each input-feeding cycle.

1.3.3 Direct Optimization

Another type of learning or training scheme is the direct learning. The adjustable parameters w are adjusted without computing gradient as in the previous learning scheme. For each iteration of simple random optimization algorithm [19], the current w are perturbed by δw, which is generated by some random scheme. If $E(w + \delta w) < E(w)$ then $w + \delta w$ is accepted to be new w, otherwise try to generate new δw again and this process is continue until $E(w)$ is in the acceptable range. A well known technique was proposed by Solis and Wets [20]. Baba [21] adapted this technique for neural networks training and succeeded.

1.4 Organization of the Thesis

The rest of this thesis is divided into 4 chapters. Chapter 2 provides the fault immunization measure as well as proofs of the several related theorems. Chapter 3 provides a modified random optimization algorithm, which will be used for relocating a weight vector to a more proper location. Chapter 4 shows the experimental results from three classification problems and discusses several related issues. Chapter 5 concludes the thesis.