行政院國家科學委員會專題研究計畫 成果報告

容錯類神經網路的適性型訓練及刪減演算法
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 95-2221-E-040-009-
執 行 期 間 ：95 年 08 月 01 日至 96 年 07 月 31 日
執 行 單 位 ：中山醫學大學資訊管理學系

計畫主持人：沈培輝

計畫參與人員：大學生-兼任助理：Ken Luo、Ben Chen

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 96 年 11 月 26 日

NSC 95-2221-E-040-009
Towards an objective function based framework for fault tolerant learning

John Sum
Institute of Electronic Commerce, National Chung Hsing University
250 Kuo Kuang Road, Taichung, Taiwan 40227
pfsum@nchu.edu.tw

***Abstract*— While conventional learning theory focus on training a fault free neural network model, fault tolerant learning aims at training a neural network that is able to tolerate anticipated fault. This paper presents a survey on the previous work done for fault tolerant neural network and proposes an objective function based framework for fault tolerant learning. In accordance with the objective functions derived for different types of network faults, algorithms for attaining a good fault tolerant neural network have thus been developed. By comparing those objective functions for fault tolerant learning with weight decay, it is found that training by adding weight decay can also improve the fault tolerance of a neural network.**

Keywords : Fault Tolerance, KL Divergence, Learning Theory

I. Introduction

In conventional learning theory, the primary objective of a learning algorithm is to attain a neural network (NN) of least mean prediction error, i.e. good generalization. To accomplish this, one approach is by the idea of adding regularizer [28], [27], [33], [34] to penalize the weights' magnitude. Another approach is by the idea of pruning [20], [25], [29], [27], [44], [39]. In which a NN is trained by a learning algorithm, and then redundant weights are identified and removed. The purposes of weight penalization and redundant weights removal are essential the same – to reduce the complexity of a NN. In accordance with the statistical learning theory [1] over-complexity can always lead to poor generalization (over-fit). Therefore, one can see that the primary focus in conventional learning theory is to seek for a NN that is of minimal complexity.

All these theories apply well to any problem, if the trained NN is hard-coded in an application software that is running in a computer. How about the trained NN is needed to be embedded in a digital hardware, like FPGA, for real time application. Component failure, low precision floating point representation and thermal noise will then affect the actual implementation of such a trained NN. The discrepancy between the hardware implemented NN and its computer simulated counterpart will lead to different types of faults to the network, such as accidentally node die, weight perturbations and etc. All these faults can also affect the performance of the *implementation* of a NN.

Consider that a trained NN has gone through the pruning step. All its redundant nodes must have been removed. Any one of reminding node is important to contribute to the output of the network. Then, the resultant network model is implemented in an FPGA. Imagine that one node is accidentally death, due to component failure. No doubt, the performance of the implemented NN will be drastically degraded.

This phenomena has been mentioned in many papers, such as in [41], [51]. A NN of good generalization might not be able to tolerate network fault. However, not much theoretical work has been reported in the literature relating those issues in regard to generalization and fault tolerance. Many questions are left to be answered. Let us point out a few.

- In conventional learning, training a NN is determined by an objective function which the learning algorithm apply. For fault tolerant learning, not all existing algorithms are defined based upon objective functions. Some of them are designed by heuristic. Is it possible to find the objective functions for them ?
- Algorithm like weight decay used to be applied in training a NN of good generalization has also been applied in training a NN of good fault tolerance. Does it mean that weight decay should be an universal technique for NN learning ?
- If the objective functions are found, what are their similarities, differences and relationships with those defined in conventional learning ?

This paper initiates the first step by proposing an objective function based framework for fault tolerant learning. The purpose is to provide a partial answer to the first and the second questions. With the objective function derived for different types of fault models, comparison can be made amongst existing fault tolerant training methods and regularization-based training methods.

The rest of the paper will be organized as follows. In the next section, a background survey on the research works related to fault tolerant NNs will be elucidated. The proposed framework is presented in Section 3 to Section 5. The conclusion is presented in Section 6.

II. Background Survey

A. Research works on the analysis of FTNN

Consider a Madaline is with threshold logic output neuron, Stevenson *et al* [48] gave a comprehensive analysis on the *probability of output error* due to different type of noises, such as input and weight noise, both additive and multiplicative. For multilayer perceptron, Choi and Choi [15] from statistical

[1] Please refer to Chapter 9 in [8] and Chapter 7 in [21]

TABLE I

RESEARCH WORKS ON THE ANALYSIS OF A FAULT TOLERANT NN.

Ref.	Fault	NN	Work
[48]	Any weight noise	Madaline	Probility of output error
[15]	Any noise	Any	Output sensitivity measure
[43]	Any noise	Madaline	Precision requirement
[10]	Mul. weight noise	RBF	Generalizaton ability
[54]	Any noise	RBF	Output sensitivity matrix
[2]	Any weight noise	MLP	Output sensitivity measure
[41]	-	-	Relationship between FT, generalization and VC dim.
[4]	Any weight noise	MLP	Generalization ability
[19]	Any weight noise	FN[a]	Error sensitivity measure

[a] Functional net

sensitivity approach to derive different *output sensitivity measures* of a network due to different type of noise. Consider a Madaline is with sigmoidal output neuron, Piche in [43] followed an approach from signal to noise ratio (SNR) and came up with a set of measures for the output sensitivity of a network with respect to different noises. Using such SNR, a weight accuracy selection algorithm is developed and applied to determine the precision requirement in hardware implementation. Townsend and Tarassenko [54] considered a radial basis function (RBF) network with multiple outputs and derived the output sensitivity in matrix form for an RBF that is suffered from perturbations in input data, radial basis function centers and output weights.

As output sensitivity is just an indirect view point to understand the effect of NN due to noise, the actual effect of noise to the performance of a NN cannot be identified easily. A more practical view point to the problem is from its actual performance — the generalization ability. Catala and Parra proposed a fault tolerance parameter model and studied the performance degradation of a RBF network if the RBF centers, widths and the corresponding weights are corrupted by multiplicative noise [10]. Bernier *et al* extended from Choi & Choi statistical sensitivity approach [15] and derived the *error sensitivity measure* for MLP [2], [4], RBF network [6] Similarly, Fontenla-Romero *et al* derived the *error sensitivity measure* for functional net [19].

Noise can be harmful to a NN. But sometimes, it can be beneficial. Murray & Edwards [36] investigated and found that adding multiplicative weight noise (and other kinds of noise) during training can improve the generalization ability of a MLP. While noise during training can improve the generalization ability of a NN, Bishop [7] showed that adding small additive white noise to a NN during training is equivalent to Tikhnov regularization. Jim *et al* [24] noticed that adding multiplicative weight noise not just can improve the generalization ability, but also can improve the convergence ability in training a recurrent NN.

B. Algorithms for dealing with multiplicative weight noise

While lot of works have been done to understand the effect of noise to the network performance, various training methods aiming to improve the fault tolerant ability of a NN have been developed. Since the effect of a multiplicative weight noise is proportional to the magnitude of the associated weight, one intuitive approach is to control the magnitude of the weights to small values. Cavalieri & Mirabella in [11] have proposed a modified backpropagation learning algorithm for multilayer perceptron. In their algorithm, a weight magnitude control step has been added in each training epoch. Whenever the magnitude of a weight has reached a predefined upper limit, it will not be updated unless the update can bring its magnitude down. Consider that the noise effect can eventually be cancelled out at the output node if all the weight values are equal, Simon in [47] suggested a distributed fault tolerance learning approach for optimal interpolation net and formulated the learning as a nonlinear programming problem, in which training error is minimized subjected to an equality constraint on weight magnitude. Extended from the work done in [10], Parra and Catala in [38] demonstrated how a fault tolerant RBF network can be obtained by using a weight decay regularizer [33]. From model sensitivity point of view, Bernier *et al* developed a method called explicit regularization to attain a MLP [3], [5] or RBF network [6] that is able to tolerate multiplicative weight noise.

C. Algorithms for dealing with node fault

To deal with node fault, those learning algorithms developed can be classified into two approaches : (1) adding heuristics (random fault or network redundancy) during training and (2) formulating the training as a nonlinear optimization problem.

Adding heuristic in the training algorithm is essentially to enforce the internal representation ability of a NN distributed widely within the hidden nodes or weights. So that, no single node or single weight is particularly important and then random removal of a node or a weight will only gracefully degrade the performance of the network. For this approach, injecting random node fault alone [45], [9] or together with random node deletion and addition [14] during training are two techniques that have demonstrated succeed in attaining fault tolerance. Adding network redundancy by replicating multiple hidden layers after a NN has been well trained [18], [40] is another one. Under the same scenario, limit weight magnitude either by adding weight decay regularizer [14] or hard bounding the weight magnitude to a small value during training [11] are another two techniques that can succeed in obtaining a fault tolerant NN.

Another approach is to formulate the learning directly as a constraint optimization problem. Neti et al [37] defined the problem as a minimax problem, in which the objective function to be minimized is the maximum of the mean square errors over all fault models. Deodhare et al took a similar idea in [16] by defining the objective function to be minimized as the maximum square error, over all fault models and all training samples. As the computational cost in solving these minmax problem could be severe for large number of hidden units, Simon & El-Sherief in [46] and Phatak & Tcherner in [42] formulated the learning problem to a simpler unconstraint optimization problem, in which the objective function consists of two terms namely the mean square errors of the fault-free model and the ensemble average of the mean square errors over all fault models. Although solving unconstraint optimization problem is a lot more easy compared with a minimax problem, these formulations are still suffered from sever computational burden when their formulations are extended to handling multiple nodes fault. In view of the lacking of a theoretical framework and the difficulty in extending the existing approaches to multiple nodes fault, Leung *et al* in

TABLE II

RESEARCH WORKS ON THE ALGORITHMS DEVELOPED.

Ref.	Fault	NN	Idea
[37]	Single node fault	MLP	Minimax Problem[1]
[9], [45]	Node fault	MLP	Injecting random node fault during training
[35]	Weight noise	MLP	Adding weight noise during training
[14]	Weight noise &	MLP	Weghit decay[2]
[18], [40]	Node fault	MLP	Adding redundancy
[16]	Single node fault	MLP	Minimax problem[3]
[11]	Node fault	MLP	Weight magnitude bounding
[38]	Mul. weight noise	RBF	Apply weight decay algo.
[3], [5]	Mul. weight noise	MLP	Explicit regularization
[47]	Weight noise	IN[a]	Nonlinear program[4]
[42]	Single node fault	MLP	Nonlinear program[5]
[30]	Mul. nodes fault	RBF	Fault tolerant regularizer
[50]	Mul. weight noise	RBF	Apply KL divergence

[a] Interpolation net

[1] $\min_\theta\{\max_{\tilde\theta} 1/N\sum_{k=1}^N (y_k - f(x_k,\tilde\theta|\theta))^2\}$

[2] Apply weight decay algorithm with random node fault injection during training

[3] $\min_\theta\{\max_{\tilde\theta}\max_k (y_k - f(x_k,\tilde\theta|\theta))^2\}$

[4] Minimizing training error subject to equality constraint on weight magnitude

[5] $1/N\sum_{k=1}^N (y_k - f(x_k,\theta))^2 + \alpha|\Omega_{\tilde\theta}|^{-1}\sum_{\tilde\theta\in\Omega_{\tilde\theta}} 1/N\sum_{k=1}^N (y_k - f(x_k,\tilde\theta|\theta))^2$

[30] and Sum in [51] have attempted to these problems by devising an objective function for fault tolerant learning.

III. OBJECTIVE FUNCTION BASED FRAMEWORK

A. Notations

Let $\mathcal{M}_0$ be the unknown system to be modeled. The input and output of $\mathcal{M}_0$ are denoted by x and y respectively. The only information we know about $\mathcal{M}_0$ is a set of measurement data $\mathcal{D}$, where $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^N$. Making use of this data set, an estimated model $\hat{\mathcal{M}}$ that is *good* enough to capture the *general behavior* of the unknown system can be obtained. For many real-time applications, this *good* model $\hat{\mathcal{M}}$ will furthermore be mapped onto a hardware implementation, like FPGA or DSP chip. As it is known that a hardware implementation of a model $\hat{\mathcal{M}}$ can never be perfect. We denote this inaccurate implementation of $\hat{\mathcal{M}}$ by $\tilde{\mathcal{M}}$. The conceptual difference amongst $\mathcal{M}_0$, $\hat{\mathcal{M}}$ $\tilde{\mathcal{M}}$ is shown in Figure 1. Finally, we let Ω be the set of models in which $\hat{\mathcal{M}}$ and $\tilde{\mathcal{M}}$ are defined.

In conventional learning theory, it is assumed that the implementation of a model $\mathcal{M}_0$ is fault-free. Therefore $\tilde{\mathcal{M}}$ is equal to $\hat{\mathcal{M}}$. In such case, the learning algorithm for obtaining the best implemented model is basically the same as the learning algorithm for obtaining the best estimated model.

In FTL, such assumption is not existed. An implementation of a model $\mathcal{M}_0$, denoted by $\tilde{\mathcal{M}}$, is defined as a random model probabilistically depended on the model $\mathcal{M}$. The set of models in which $\tilde{\mathcal{M}}$ can be defined is denoted by $\tilde{\Omega}_{\mathcal{M}}$. Clearly, $\tilde{\Omega}_{\mathcal{M}} \subset \Omega$. The conditional probability is denoted by $P(\tilde{\mathcal{M}}|\hat{\mathcal{M}})$, which is depended on the property of the fault model concerned. It could be very complicated if multiple fault models co-exist.

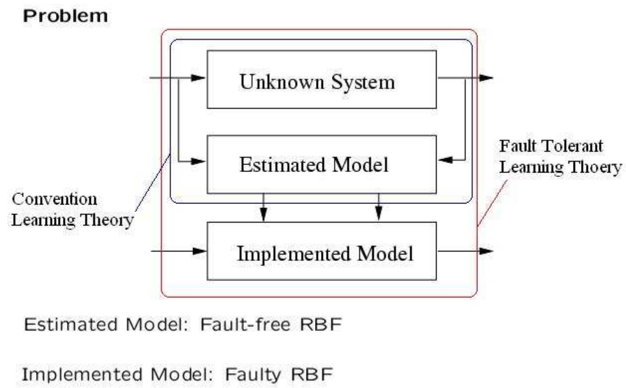

Fig. 1. Framework of fault tolerant learning.

B. Measure $\mathcal{L}(\mathcal{M}|\mathcal{D})$

To search for the best model $\hat{\mathcal{M}}$, one would need to define a measure to evaluate the closeness between two models. In convention learning, *generalization ability* and *a posterior* probability are two common measures being applied to measure the closeness between a model $\mathcal{M}$ and the unknown model $\mathcal{M}_0$.

a) Estimation: For a set of data $\mathcal{D}$ and let $J(\mathcal{M}|\mathcal{D})$ be the measure, the best *estimated model* $\hat{\mathcal{M}}$ will be defined by

$$\hat{\mathcal{M}} = \arg\min_{\mathcal{M}\in\Omega_{\mathcal{M}}} \{J(\mathcal{M}|\mathcal{D})\}. \quad (1)$$

b) Implementation: While in FTL, the focus is on the implemented model. The best *implemented model* $\hat{\mathcal{M}}_I$ is defined as the one minimizing the expectation of $J(\mathcal{M}|\mathcal{D})$ over Ω.

$$\mathcal{L}(\mathcal{M}|\mathcal{D}) = \int_{\tilde{\mathcal{M}}\in\tilde{\Omega}_{\mathcal{M}}} J(\tilde{\mathcal{M}}|\mathcal{D})P(\tilde{\mathcal{M}}|\mathcal{M})d\tilde{\mathcal{M}}. \quad (2)$$

$$\hat{\mathcal{M}}_I = \arg\min_{\mathcal{M}\in\Omega_{\mathcal{M}}} \{\mathcal{L}(\mathcal{M}|\mathcal{D})\}. \quad (3)$$

The learning algorithm that can search for the $\mathcal{M}_I$ is called a *fault tolerant learning algorithm*.

IV. ESTIMATED MODELS Ω

To clarify the concept ideas about estimated model set, let us take RBF networks as an example. Consider the estimated model is an RBF network consisting of M hidden nodes. In which only the output weights can be tunable but the basis centers and widths are fixed, an RBF network can be formulated as

$$\sum_{i=1}^M \theta_i\phi_i(x),$$

where $\phi_i(x)$ for all $i = 1, 2, \cdots, M$ are the radial basis functions given by

$$\phi_i(x) = \exp\left(-\frac{(x-c_i)^2}{\sigma}\right), \quad (4)$$

c_is are the radial basis function centers and the positive parameter $\sigma > 0$ controls the width of the radial basis functions.

For $k = 1, 2, \cdots, N$

$$\mathcal{M}_0 \quad : \quad y_k = f(x_k) + e_k, \tag{5}$$

where (x_k, y_k) is the k^{th} input-output pair that is measured from an unknown deterministic system $f(x)$ with random output noise e_k. To model the unknown system, we assume that $f(x)$ can be realized by an RBF network, i.e.

$$\begin{aligned} \mathcal{M} \quad : \quad y_k &= \sum_{i=1}^{M} \theta_i \phi_i(x_k) + e_k & (6) \\ e_k &\sim \mathcal{N}(0, S_e), & (7) \end{aligned}$$

for all $k = 1, 2, \cdots, N$. S_e is known in advance, a model $\mathcal{M}$ in Ω can indeed be represented by an M-vector,

$$\theta = (\theta_1, \theta_2, \cdots, \theta_M)^T.$$

The model set Ω is isomorphic to an M-dimension Euclidean space, R^M.

The best estimated model $\hat{\mathcal{M}}$ is thus represented $\hat{\theta}$. Equation (1) is rewritten as follows :

$$\hat{\theta} = \arg \min_{\theta \in R^M} \left\{ J(\theta|\mathcal{D}) \right\}. \tag{8}$$

Here $J(\theta|\mathcal{D})$ can be defined in one of the following forms.

1) Sum Square Errors (SSE) :

$$J(\theta|\mathcal{D}) = \frac{1}{N} \sum_{k=1}^{N} (y_k - f(x_k, \theta))^2. \tag{9}$$

2) SSE with Regularizer (Weight Decay) :

$$J(\theta|\mathcal{D}) = \frac{1}{N} \sum_{k=1}^{N} (y_k - f(x_k, \theta))^2 + \lambda \theta^T \theta, \quad (\lambda > 0). \tag{10}$$

3) Likelihood Probability :

$$J(\theta|\mathcal{D}) = -P(\mathcal{D}|\theta). \tag{11}$$

4) Log Likelihood :

$$J(\theta|\mathcal{D}) = -\log P(\mathcal{D}|\theta). \tag{12}$$

5) *A Posterior* Probability :

$$J(\theta|\mathcal{D}) = -\frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}. \tag{13}$$

6) Log *A Posterior* Probability :

$$J(\theta|\mathcal{D}) = -\log P(\mathcal{D}|\theta) - \log P(\theta). \tag{14}$$

The probability $P(\theta)$ which appears in Equation (13) and Equation (14) is the *A Prior* distribution of θ.

The best implemented model $\hat{\mathcal{M}}_I$ is thus represented $\hat{\theta}_I$. Equation (2) can be rewritten as follows :

$$\begin{aligned} \mathcal{L}(\theta|\mathcal{D}) &= \int_{\tilde{\theta} \in \tilde{\Omega}_\theta} J(\tilde{\theta}|\mathcal{D}) P(\tilde{\theta}|\theta) d\tilde{\theta} & (15) \\ \hat{\theta}_I &= \arg \min_{\theta \in R^M} \left\{ \mathcal{L}(\theta|\mathcal{D}) \right\}. & (16) \end{aligned}$$

The integration is taken over the R^M space. The probability $P(\tilde{\theta}|\theta)$ is depended on the fault model concerned. Note that this probability is not the same as the *A Prior* probability $P(\theta)$.

If there are only finite number of possible faulty models, the objective function defined in Equation (15) would be given by

$$\mathcal{L}(\theta|\mathcal{D}) = \sum_{\tilde{\theta} \in \tilde{\Omega}_\theta} J(\tilde{\theta}|\mathcal{D}) P(\tilde{\theta}|\theta) d\tilde{\theta}. \tag{17}$$

The set of faulty models is depended on the estimated model θ.

One should note that the best estimated model (i.e. the fault-free model) obtained either by Equation (11) or Equation (12) are the same because

$$\arg \min_{\theta \in R^M} \left\{ -P(\mathcal{D}|\theta) \right\} = \arg \min_{\theta \in R^M} \left\{ -\log P(\mathcal{D}|\theta) \right\}.$$

However, for fault tolerant cases, there will have no such guarantee that

$$\arg \min_{\theta \in R^M} \left\{ -\int P(\mathcal{D}|\tilde{\theta}) P(\tilde{\theta}|\theta) d\tilde{\theta} \right\}$$

is the same as

$$\arg \min_{\theta \in R^M} \left\{ \int \left(-\log P(\mathcal{D}|\tilde{\theta}) \right) P(\tilde{\theta}|\theta) d\tilde{\theta} \right\}.$$

The same reason applies to Equation (13) and Equation (14).

Apart from defining an RBF network as in Equation (7), one can also define the estimated model in other forms. For instance,

$$\begin{aligned} y_k &= \theta_0 + \sum_{i=1}^{M} \theta_i \phi_i(x_k) + e_k, & (18) \\ e_k &\sim \mathcal{N}(0, S_e), & (19) \end{aligned}$$

for $k = 1, 2, \cdots, N$. For a given S_e, the estimated model set will be isomorphic to the R^{M+1} space.

If we assume that the values of c_is and σ in the M basis functions are not predefined, an RBF model will be parameterized by an $(2M + 2)$-vector,

$$(\theta_0, \theta_1, \cdots, \theta_M, c_1, c_2, \cdots, c_M, \sigma)$$

The estimated model set Ω will thus be isomorphic to the R^{2M+2} space.

V. Implemented Models $\tilde{\Omega}_\mathcal{M}$

Recall that an implemented model of $\mathcal{M}$ is a model, in which part of its structure is faulty. In this section, three typical fault models will be introduced including (1) the multiplicative weight noise (2) single-node fault and (3) multiple-nodes fault. Similarly, we use RBF network as an example for illustration.

A. Multiplicative weight noise with $J(\theta|\mathcal{D}) = SSE$

Mulplicative weight noise exists whenever a weight is encoded in a low precision binary form. In order not to divert the focus of this section, the exaplination of this effect is presented in Appendix A.

Using the model described in Equation (55), an implementation of a model θ (denoted by $\tilde{\theta}$) can be defined as follows :

$$\begin{aligned} \tilde{\theta}_i &= \theta_i + \beta_i \, \theta_i, & (20) \\ \beta_i &\sim \mathcal{N}(0, S_\beta), & (21) \end{aligned}$$

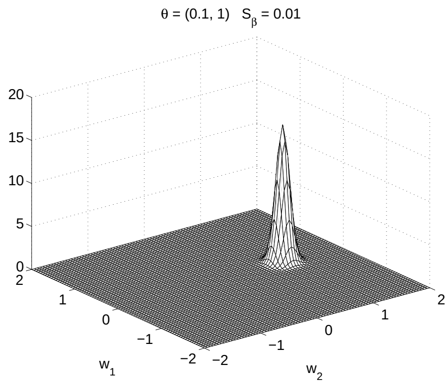

Fig. 2. For multiplicative weight noise case, the conditional probability $P(\tilde{\theta}|\theta)$ for θ equals to $(0.1, 1)^T$.

for all $i = 1, 2, \cdots, M$. In other word,

$$P(\beta_i) = \frac{1}{\sqrt{2\pi S_\beta}} \exp\left(-\frac{\beta_i^2}{2S_\beta}\right) \quad \forall\, i = 1, \cdots, M. \qquad (22)$$

Let $\theta = (\theta_1, \theta_2, \cdots, \theta_M)^T$ and $\beta = (\beta_1, \beta_2, \cdots, \beta_M)^T$,

$$\begin{aligned} \tilde{\theta} &= \theta + A(\theta)\beta, \\ A(\theta) &= \textbf{diag}\{\theta_1, \theta_2, \cdots, \theta_M\}. \end{aligned}$$

So,

$$P(\tilde{\theta}|\theta) \sim \mathcal{N}(\theta, S_\beta A^2(\theta)). \qquad (23)$$

An example of $P(\tilde{\theta}|\theta)$ is shown in Figure 2. Here $\theta = (0.1, 1)^T$ and the weight noise variance S_β is 0.01.

One should note that $\theta, \tilde{\theta} \in R^M$, and $\tilde{\Omega}_\theta = \Omega = R^M$. For $J(\theta|\mathcal{D})$ is sum square errors,

$$\mathcal{L}(\theta|\mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N}\int_{\tilde{\theta}\in\tilde{\Omega}} (y_k - f(x_k, \tilde{\theta}))^2 P(\tilde{\theta}|\theta) d\tilde{\theta}. \qquad (24)$$

Consider the transition probability $P(\tilde{\theta}|\theta)$ as defined in Equation (23), it can be reduced to the following explicit regularization form [3].

$$\mathcal{L}(\theta|\mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N}(y_k - f(x_k, \theta))^2 + S_\beta \theta^T \left[\frac{1}{N}\sum_{k=1}^{N}\mathbf{G}(x_k)\right]\theta, \qquad (25)$$

where $\mathbf{G}(x_k)$ is a diagonal matrix defined as follows :

$$\mathbf{G}(x_k) = \textbf{diag}\left\{\phi_1^2(x_k), \phi_2^2(x_k), \cdots, \phi_M^2(x_k)\right\}. \qquad (26)$$

For RBF network with predefined basis function centers and widths, $\hat{\theta}_I$ is given by

$$\hat{\theta}_I = (H_\phi + S_\beta Q_g)^{-1}\left(\frac{1}{N}\sum_{k=1}^{N} y_k \phi(x_k)\right), \qquad (27)$$

where

$$\begin{aligned} H_\phi &= \frac{1}{N}\sum_{k=1}^{N}\phi(x_k)\phi^T(x_k) \\ Q_g &= \begin{bmatrix} g_1 & 0 & \cdots & 0 \\ 0 & g_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & g_M \end{bmatrix} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{G}(x_k). \end{aligned}$$

It is clear that, those $\tilde{\theta}$s with high probability are clustered around θ. If we restrict the $\tilde{\theta}$ only those with $P(\tilde{\theta}|\theta)$ larger than a small positive number δ, the best implemented model can be re-defined as follows :

$$\begin{aligned} \mathcal{L}^r(\theta|\mathcal{D}) &= \int_{\tilde{\theta}\in\tilde{\Omega}_\theta^r} J(\tilde{\theta}|\mathcal{D})P(\tilde{\theta}|\theta)d\tilde{\theta} & (28) \\ \hat{\theta}_I &= \arg\min_{\theta\in R^M}\{\mathcal{L}^r(\theta|\mathcal{D})\}, & (29) \end{aligned}$$

where $\tilde{\Omega}_\theta^r = \{\tilde{\theta}|P(\tilde{\theta}|\theta) \geq \delta\}$. The computation complexity for $\hat{\theta}_I$ can be largely reduced. This is particularly advantageous when the dimension of θ is large.

B. Multiplicative weight noise with $J(\theta|\mathcal{D}) = -\log P(\mathcal{D}|\theta)$

For RBF, $P(y_k|x_k, \beta, \theta)$ is given by

$$\frac{1}{\sqrt{2\pi S_e}} \exp\left(-\frac{(y_k - \sum_{i=1}^{M}\phi_i(x_k)(1+\beta_i)\theta_i)^2}{2S_e}\right) \qquad (30)$$

for all $k = 1, 2, \cdots, N$. Putting the definitions of $P(\beta_i)$ in Equation (22) and $P(y|x, \beta, \theta)$ in Equation (30), and integrate over all possible β, we have the distribution

$$\begin{aligned} & P(y_k|x_k, \theta) \\ = & \int P(y_k|x_k, \beta, \theta)P(\beta)d\beta \\ = & \frac{1}{\sqrt{2\pi S(x_k, \theta)}} \exp\left(-\frac{(y_k - \phi^T(x_k)\theta)^2}{2S(x_k, \theta)}\right) \quad (31) \end{aligned}$$

for all $k = 1, 2, \cdots, N$.

$$\begin{aligned} S(x, \theta) &= S_e + S_\beta \phi^T(x)A^2(\theta)\phi(x) & (32) \\ &= S_e + S_\beta \sum_{i=1}^{M}\phi_i^2(x)\theta_i^2. & (33) \end{aligned}$$

The likelihood probability will be given as follows :

$$\begin{aligned} P(\mathcal{D}|\theta) &= \prod_{k=1}^{N}\int P(y_k|x_k, \tilde{\theta}, \theta)P(\tilde{\theta}|\theta)d\tilde{\theta} & (34) \\ &= \prod_{k=1}^{N}\int P(y_k|x_k, \beta, \theta)P(\beta)d\beta. & (35) \end{aligned}$$

The $\mathcal{L}(\theta|\mathcal{D})$ can then be written as follows :

$$\begin{aligned} \mathcal{L}(\theta|\mathcal{D}) &= -\sum_{k=1}^{N}\log\int P(y_k|x_k, \beta, \theta)P(\beta)d\beta & (36) \\ &= \frac{1}{2}\log 2\pi + \frac{1}{2N}\sum_{k=1}^{N}\log S(x_k, \theta) \\ &\quad + \frac{1}{2N}\sum_{k=1}^{N}\frac{(y_k - \phi^T(x_k)\theta)^2}{S(x_k, \theta)}. & (37) \end{aligned}$$

Hence, $\hat{\theta}_I$ can be obtained by

$$\arg\min_\theta\left\{\frac{1}{2N}\sum_{k=1}^{N}\log S(x_k, \theta) + \frac{1}{2N}\sum_{k=1}^{N}\frac{(y_k - \phi^T(x_k)\theta)^2}{S(x_k, \theta)}\right\}. \qquad (38)$$

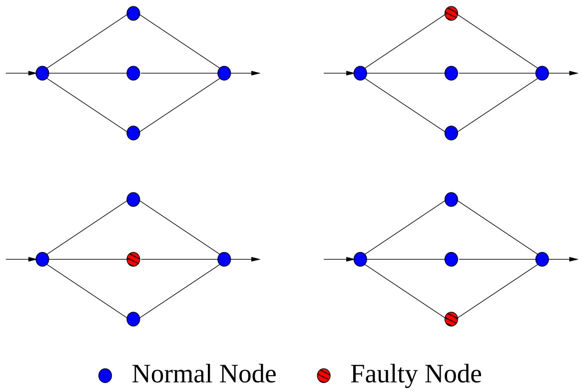

Fig. 3. Single-node fault NN models. For a network of M hidden nodes, there are M possible single-node fault models.

By using the idea of gradient descent, a training algorithm can thus be derived. Taking the gradients of the 2nd and the 3rd terms in Equation (37), it is readily obtained

$$\frac{\partial}{\partial\theta}\log S(x_k,\theta) = \frac{2S_\beta}{S(x_k,\theta)}\mathbf{G}(x_k)\theta, \tag{39}$$

$$\begin{aligned}\frac{\partial}{\partial\theta}\frac{(y_k-\phi^T(x_k)\theta)^2}{S(x_k,\theta)} &= -\frac{2S_\beta(y_k-\phi^T(x_k)\theta)^2}{S^2(x_k,\theta)}\mathbf{G}(x_k)\theta \\ &\quad -\frac{2(y_k-\phi^T(x_k)\theta)}{S(x_k,\theta)}\phi(x_k), \end{aligned}\tag{40}$$

where $\mathbf{G}(x_k)$ is a diagonal matrix defined as in Equation (26).

A fault tolerant RBF network can thus be obtained by the following gradient descent algorithm :

$$\theta(t+1) = \theta(t) - \mu\frac{\partial}{\partial\theta}\mathcal{L}(\theta(t)|\mathcal{D}), \tag{41}$$

where μ is a small positive value corresponding to the step size and

$$\begin{aligned}\frac{\partial\mathcal{L}(\theta|\mathcal{D})}{\partial\theta} &= \frac{S_\beta}{N}\sum_{k=1}^N\left(\frac{1}{S(x_k,\theta)} - \frac{(y_k-\phi^T(x_k)\theta)^2}{S^2(x_k,\theta)}\right)\mathbf{G}(x_k)\theta \\ &\quad -\frac{1}{N}\sum_{k=1}^N\frac{(y_k-\phi^T(x_k)\theta)}{S(x_k,\theta)}\phi(x_k).\end{aligned}\tag{42}$$

The initial condition $\theta(0)$ is set to be a small random vector close to null.

C. Single node fault with $J(\theta|\mathcal{D}) = SSE$

Once a node has been faulty, we assume that its output will be stuck at zero. Therefore, an RBF network with its i^{th} node being faulty will be denoted by an M-vector θ_{-i}, which is identical to θ except that the i^{th} element is zero.

$$\theta_{-i} = (\theta_1,\theta_2,\cdots,\theta_{i-1},0,\theta_{i+1},\cdots,\theta_M)^T$$

Assume that *there is at most one node will be removed randomly*. The probability that a network will be faulty is q. Once a network is faulty, there is uniformly random for any one of the node is fault, Figure 3. Under such circumstance,

$$\Omega = R^M, \tag{43}$$

$$\tilde{\Omega}_\theta = \{\theta,\theta_{-1},\theta_{-2},\cdots,\theta_{-M}\}. \tag{44}$$

A node will be fault is about q/M probability.

$$P(\tilde{\theta}|\theta) = \begin{cases} 1-q & \text{if } \tilde{\theta}=\theta \\ q/M & \text{if } \tilde{\theta}=\theta_{-1} \\ \vdots & \vdots \\ q/M & \text{if } \tilde{\theta}=\theta_{-M}. \end{cases}\tag{45}$$

For $J(\theta|\mathcal{D})$ is defined as the sum square errors,

$$\mathcal{L}(\theta|\mathcal{D}) = (1-q)J(\theta|\mathcal{D}) + \frac{q}{M}\sum_{i=1}^M J(\theta_{-i}|\mathcal{D}). \tag{46}$$

In which,

$$\begin{aligned}J(\theta_{-i}|\mathcal{D}) &= J(\theta|\mathcal{D}) + \theta_i^2 g_i \\ &\quad + 2\theta_i\frac{1}{N}\sum_{k=1}^N(y_k-\phi^T(x_k)\theta)\phi_i(x_k)\end{aligned}\tag{47}$$

where g_i is the i^{th} diagonal element of Q_g. Hence, the objective function for attaining a RBF network to tolerate single node fault can be written as follows :

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{D}) &= J(\theta|\mathcal{D}) + \frac{2q}{M}\frac{1}{N}\sum_{k=1}^N y_k\phi^T(x_k)\theta \\ &\quad + \frac{q}{M}\theta^T[Q_g - 2H_\phi]\theta.\end{aligned}\tag{48}$$

Taking the derivative of $\mathcal{L}(\theta|\mathcal{D})$ and setting it to zero, $\hat{\theta}_I$ can be obtained as follows :

$$\hat{\theta}_I = \left(H_\phi + \frac{q/M}{1-q/M}Q_g\right)^{-1}\frac{1}{N}\sum_{k=1}^N y_k\phi(x_k). \tag{49}$$

The matrix $\frac{q/M}{1-q/M}Q_g$ which appears in the last equation plays a role similar to a regularizer.

D. Multiple nodes fault with $J(\theta|\mathcal{D}) = SSE$

We assume that a node fault is equivalent to permanently set the output of the node zero. Therefore, a faulty RBF $\hat{f}(x,\tilde{\theta})$, where $\tilde{\theta} = (\tilde{\theta}_1,\tilde{\theta}_2,\cdots,\tilde{\theta}_M)^T$ and

$$\tilde{\theta}_i = \beta_i\theta_i, \tag{50}$$

could be defined by multiplying each $\phi_i(x)$ by a random binary variable β_i :

$$f(x,\theta,\beta) = \sum_{i=1}^M \beta_i\theta_i\phi_i(x). \tag{51}$$

When $\beta_i = 1$, the i^{th} node is normal. When $\beta_i = 0$, the i^{th} node is fault. We assume that all nodes are of equal fault rate p, i.e.

$$P(\beta_i) = \begin{cases} p & \text{if } \beta_i = 0 \\ 1-p & \text{if } \beta_i = 1. \end{cases}\tag{52}$$

for $i = 1,2,\cdots,M$ and $\beta_1,\cdots,\beta_M$ are independent random variables.

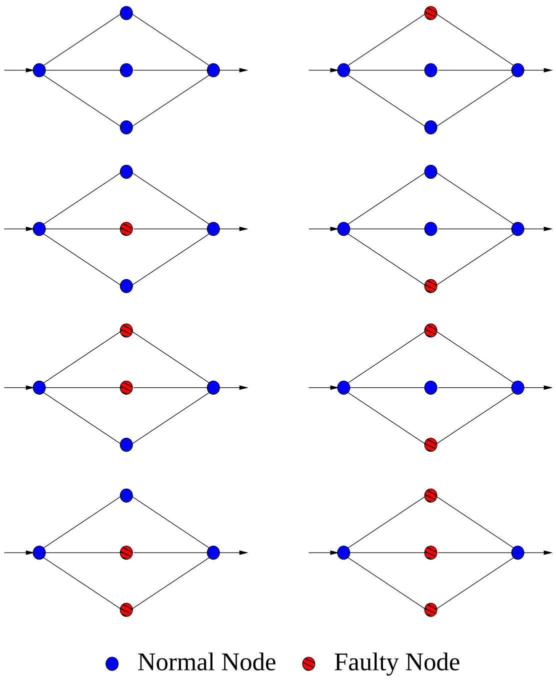

Fig. 4. Multiple-nodes fault NN models. For a network of n hidden nodes, there are $2^n - 1$ possible multiple-nodes fault models.

The objective function for attaining an optimal fault tolerant RBF against multiple nodes fault with fault rate p is given by

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{D}) &= \frac{1}{N}\sum_{k=1}^{N} y_k^2 - 2(1-p)\frac{1}{N}\sum_{k=1}^{N} y_k \phi^T(x_k)\theta \\ &\quad +(1-p)\theta^T\left\{(1-p)H_\phi + pQ_g\right\}\theta.\end{aligned}$$

The implicit regularizer is given by $p\theta^T(Q_g - H_\phi)\theta$.

Taking derivative the $\mathcal{L}(\theta|\mathcal{D})$ with respect to θ and setting it to zero, $\hat{\theta}_I$ can be obtained as follows :

$$\hat{\theta} = \left(H_\phi + p\left(Q_g - H_\phi\right)\right)^{-1}\frac{1}{N}\sum_{k=1}^{N} y_k\phi(x_k). \qquad (53)$$

Observe that $\hat{\theta}$ above is also the solution of

$$\mathcal{L}(\theta|\mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - \phi^T(x_k)\theta\right)^2 + \theta^T\Sigma\theta, \qquad (54)$$

where $\Sigma = p(Q_g - H_\phi)$, minimizing $\mathcal{L}(\theta|\mathcal{D})$ is equivalent to minimizing the mean square training errors $N^{-1}\sum_{k=1}^{N}\left(y_k - \phi^T(x_k)\theta\right)^2$ plus an additional regularizer term $\theta^T\Sigma\theta$.

VI. Conclusion

In this paper, a survey on fault tolerant NN researches has been elucidated. Then, an objective function based framework is proposed. Using RBF as an example, four objective functions for dealing with three different types of fault models have been derived. In sequel, four fault tolerant learning algorithms have been developed. By comparing the equations for the best implemented models, $\hat{\theta}_I$, in dealing with multiplicative weight noise and single node fault, it is able to explain why training an RBF by adding weight decay can also improve the fault tolerance.

Finally, it should be noted that investigation the fault tolerance of NNs has still been a valuable problem in the NN community [12], [13], [17], [52], [53]. The framework developed in this paper is just in its preliminary stage. Further work should have to do in order to make it complete and connect to the conventional NN learning theory.

Contributed Publications

The work is supported by a research grant from the Taiwan National Science Council under project number NSC 95-2221-E-040-009. A number of papers have been published in accordance with the grant supported.

- **John Sum** and Chi-sing Leung, Prediction error of a fault tolerant neural network, *Neurocomputing*. (SCI) (Revised and resubmitted)
- **John Sum**, Chi-sing Leung and Kevin Ho, On objective function, regularizer and prediction error of a learning algorithm for dealing with multiplicative weight noise, *IEEE Transactions on Neural Networks*. (SCI) (Revised and resubmitted)
- **John Sum**, Towards an objective function based framework for fault tolerant learning, in *Proc. TAAI'2007*.
- **J. Sum**, C.S. Leung, L.P. Hsu, Y.F. Huang, An objective function for single node fault RBF learning, in *Proc. TAAI'2007*.
- **John Sum**, Chi-sing Leung and Lipin Hsu, Fault tolerant learning using Kullback-Leibler Divergence, to appear in *Proc. TENCON'2007* Taipei, 2007.

References

[1] Shun'ichi Amari, *Differential-geometrical methods in statistics*, Lecture notes in statistics, Springer-Verlag, Berlin, 1985.

[2] Bernier J.L. *et al*, An accurate measure for multiplayer perceptron tolerance to weight deviations, *Neural Processing Letters*, Vol.10(2), 121-130, 1999.

[3] Bernier J.L. *et al*, Obtaining fault tolerance multilayer perceptrons using an explicit regularization, *Neural Processing Letters*, Vol.12, 107-113, 2000.

[4] Bernier J.L. *et al*, A quantitative study of fault tolerance, noise immunity and generalization ability of MLPs, *Neural Computation*, Vol.12, 2941-2964, 2000.

[5] Bernier J.L. *et al* Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations, *Neurocomputing*, Vol.31, 87-103, 2000.

[6] Bernier J.L. *et al*, Assessing the noise immunity and generalization of radial basis function networks, *Neural Processing Letter*, Vol.18(1), 35-48, 2003.

[7] Bishop C.M., Training with noise is equivalent to Tikhnov regularization, *Neural Computation*, Vol.7, 108-116, 1995.

[8] Bishop C.M., *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[9] Bolt G., *Fault tolerant in multi-layer Perceptrons*. PhD Thesis, University of York, UK, 1992.

[10] Catala M. A. and X.L. Parra, Fault tolerance parameter model of radial basis function networks, *IEEE ICNN'96*, Vol.2, 1384-1389, 1996.

[11] Cavalieri S. and O. Mirabella, A novel learning algorithm which improves the partial fault tolerance of multilayer NNs, *Neural Networks*, Vol.12, 91-106, 1999.

[12] Chandra P. and Y. Singh, Fault tolerance of feedforward artificial neural networks – A framework of study, *Proceedings of IJCNN'03* Vol.1 489-494, 2003.

[13] Chandra P. and Y. Singh, Feedforward sigmoidal networks - equicontinuity and fault-tolerance properties, *IEEE Transactions on Neural Networks*, Vol.15(6), 1350-1366, 2004.

[14] Chiu C.T. *et al.*, Modifying training algorithms for improved fault tolerance, ICNN'94 Vol.I, 333-338, 1994.

[15] Choi J.Y. and C.H. Choi, Sensitivity of multilayer perceptrons with differentiable activation functions, *IEEE Transactions on Neural Networks*, Vol. 3, 101-107, 1992.
[16] Deodhare D., M. Vidyasagar and S. Sathiya Keerthi, Sythesis of fault-tolerant feedforward neural networks using minimax optimization, *IEEE Transactions on Neural Networks*, Vol.9(5), 891-900, 1998.
[17] Eickhoff R. and U. Riickett, Tolerance of radial basis functions against stuck-at-faults, *Proc. ICANN 2005*, LNCS 3697 Springer, 1003-1008, 2005.
[18] Emmerson M.D. and R.I. Damper, Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application, *IEEE Transactions on Neural Networks*, Vol.4, 788-793, 1993.
[19] Fontenla-Romero O. *et al*, A measure of fault tolerance for functional networks, *Neurocomputing*, Vol.62, 327-347, 2004.
[20] Hassibi B and D.G. Stork (1993). Second order derivatives for network pruning: Optimal brain surgeon. In Hanson *et al.* (eds) *Advances in Neural Information Processing Systems*, 164-171.
[21] T. Hastie, R. Tibshirani, J. H. Friedman *The Elements of Statistical Learning*, Springer, 2001.
[22] S. Haykin, *Neural networks: A comprehensive foundation*, Macmillan College Publishing Company, Inc. 1994.
[23] Holt J. and J. Hwang, Finite precision error analysis of neural networks hardware implementation, *IEEE Transactions on Computers*, Vol.42, p.281-290, 1993.
[24] Jim K.C., C.L. Giles and B.G. Horne, An analysis of noise in recurrent neural networks: Convergence and generalization, *IEEE Transactions on Neural Networks*, Vol.7, 1424-1438, 1996.
[25] LeCun Y. *et al.* (1990). Optimal brain damage, *Advances in Neural Information Processing Systems 2* (D.S. Touretsky, ed.) 396-404.
[26] Kullback S., *Information Theory and Statistics*, Wiley, 1959.
[27] Chi-sing Leung, Kwok-wo Wong, Pui-fai Sum and Lai-wan Chan, A pruning method for recursive least squared algorithm, Neural Networks, 14:147-174, 2001.
[28] Leung C.S., G.H. Young, J. Sum and W.K. Kan, On the regularization of forgetting recursive least square, *IEEE Transactions on Neural Networks*, Vol.10, 1842-1846, 1999.
[29] C.S.Leung, K.W.Wong, John Sum, and L.W.Chan, On-line training and pruning for RLS algorithms, Electronics Letters, Vol.32, No.23, 2152-2153, 1996.
[30] Chi-sing Leung and John Sum, Fault tolerant regularizer for multiple nodes fault RBF, accepted for publication in *IEEE Transactions on Neural Networks*.
[31] Mackay D.J.C. (1992), A Practical Bayesian Framework for Backprop Networks, *Neural Computation*, Vol.4(3) 448-472.
[32] Merchant S, G.D. Peterson, S.K. Park and S.G. Kong, FPGA implementation of evolvable block-based neural networks, *Proc. IEEE Congress on Evolutionary Computation*, Vancouver Canada, p.3129-3136, 2006.
[33] Moody J.E., Note on generalization, regularization, and architecture selection in nonlinear learning systems, *First IEEE-SP Workshop on Neural Networks for Signal Processing*, 1991.
[34] Murata N., S. Yoshizawa and S. Amari. Network information criterion–Determining the number of hidden units for an artificial neural network model, *IEEE Transactions on Neural Networks*, Vol.5(6), pp.865-872, 1994.
[35] Murray A.F. and P.J. Edwards, Synaptic weight noise during multilayer perceptron training: fault tolerance and training improvements, *IEEE Transactions on Neural Networks*, Vol.4(4), 722-725, 1993.
[36] Murray A.F. and P.J. Edwards, Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training, *IEEE Transactions on Neural Networks*, Vol.5(5), 792-802, 1994.
[37] Neti C. M.H. Schneider and E.D. Young, Maximally fault tolerance neural networks, *IEEE Transactions on Neural Networks*, Vol.3(1), 14-23, 1992.
[38] Parra X. and A. Catala, Fault tolerance in the learning algorithm of radial basis function networks, *Proc. IJCNN 2000*, Vol.3, 527-532, 2000.
[39] Pedersen M.W., L.K. Hansen and J. Larsen. Pruning with generalization based weight saliencies: γOBD, γOBS. *Advances in Information Processing Systems 8* 521-528, 1996.
[40] Phatak D.S. and I. Koren, Complete and partial fault tolerance of feedforward neural nets., *IEEE Transactions on Neural Networks*, Vol.6, 446-456, 1995.
[41] Phatak D.S., Relationship between fault tolerance, generalization and the Vapnik-Cervonenkis (VC) dimension of feedforward ANNs, *IJCNN'99*, Vol.1, 705-709, 1999.
[42] Phatak D.S. and E. Tcherner, Synthesis of fault tolerance neural networks, *Proc. IJCNN'02*, 1475-1480, 2002.
[43] Piche S.W., The selection of weight accuracies for Madalines, *IEEE Transactions on Neural Networks*, Vol.6, 432-445, 1995.
[44] Reed R., Pruning algorithms – A survey, *IEEE Transactions on Neural Networks*, Vol.4(5), 740-747, 1993.
[45] Sequin C.H. and R.D. Clay, Fault tolerance in feedforward artificial neural networks, *Neural Networks*, Vol.4, 111-141, 1991.
[46] Simon D. and H. El-Sherief, Fault-tolerance training for optimal interpolative nets, *IEEE Transactions on Neural Networks*, Vol.6, 1531-1535, 1995.
[47] Simon D., Distributed fault tolerance in optimal interpolative nets, *IEEE Transactions on Neural Networks*, Vol.12(6), 1348-1357, 2001.
[48] Stevenson M., R. Winter and B. Widrow, Sensitivity of feedfoward neural networks to weight errors, *IEEE Transactions on Neural Networks*, Vol.1, 71-80, 1990.
[49] John Sum, Chi-sing Leung and Kevin Ho, On objective function, regularizer and prediction error of a learning algorithm for dealing with multiplicative weight noise, in submission to *IEEE Transactions on Neural Networks*.
[50] John Sum and Chi-sing Leung, Derivation of explicit regularization from KL divergence, *Neural Processing Letters* (In revision).
[51] John Sum, On a multiple nodes fault tolerant training for RBF: Objective function, sensitivity analysis and relation to generalization, Proceedings of TAAI'05, Tainan, ROC, 2005.
[52] Elko B. Tchernev, Rory G. Mulvaney, and Dhananjay S. Phatak, Investigating the Fault Tolerance of Neural Networks, *Neural Computation*, Vol.17, 1646-1664, 2005.
[53] Elko B. Tchernev, Rory G. Mulvaney, and Dhananjay S. Phatak, Perfect fault tolerance of the n-k-n network, *Neural Computation*, Vol.17, 1911-1920, 2005.
[54] Townsend N.W. and L. Tarassenko, Estimations of error bounds for neural network function approximators, *IEEE Transactions on Neural Networks*, Vol.10(2), 217-230, 1999.
[55] Vollmer U. and A. Strey, Experimental study on the precision requirements of RBF, RPROP and BPTT training, *Proc. IEE Ninth International Conference on Artificial Neural Networks*, p.239-244, 1999.

APPENDIX

A. Source of Multiplicative Weight Noise

Multiplicative weight noise usually appears when a neural network is implemented by a FPGA. Owing to increase computational efficiency and reduce circuit complexity, floating point arithmetic is avoided. Each number in FPGA is encoded by a low precision (finite bits) binary number [23], [32], [55], in a form as follows $\pm rrr.pppppppppppp$. The first bit is the sign bit. Then, a decimal number 3.125 will be encoded to 1011001000000000. For an 8-bits format of the form $\pm rr.pppp$, 3.125 will be encoded to 11100100.

The beauty of this encoding scheme is that the arithmetic operations, such as $+$ and $\times$, can be accomplished by integer arithmetic. The drawback is that quantization error will exist. For example, 3.124 and 3.126 cannot be encoded perfectly. Using the 8-bits format, 3.124 and 3.126 will be encoded to 11100100, if the number is rounded to its nearest 8-bits binary number. Therefore, an error will exist between a decimal z and its encoded counterpart $\tilde{z}$.

To study the behavior of this error, we consider the 8-bits format and let $b = (z - \tilde{z})/z$. Then 10000 zs are uniformly random sampled in the range $[-4, 4]$. The histogram of the corresponding bs is plotted in Figure 5. Clearly, the distribution can be treated as a Gaussian distribution with mean zero. Hence, $\tilde{z}$ can be modeled as an random variable given by

$$\tilde{z} = z + bz, \tag{55}$$

where $b \sim \mathcal{N}(0, S_b)$. In accordance with the simulation, the value of S_b is 0.0054.

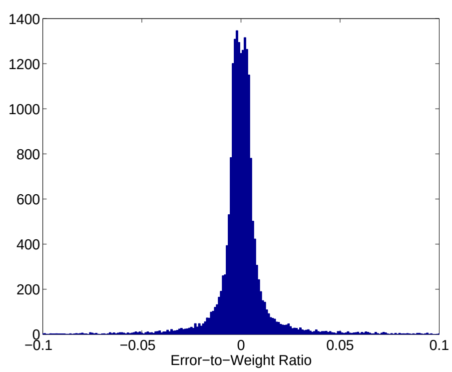

Fig. 5. Finite precision error.