Fault tolerant learning for neural networks : Survey, framework and future work

John Sum
Institute of E-Commerce, National Chung Hsing University,
250 Kuo Kuang Road, Taichung,Taiwan 40227

Abstract

While conventional learning theory focus on training a neural network to attain good generalization, fault tolerant learning aims at training a neural network to attain acceptable generalization even if network fault might appear in the future. This paper presents an extensive survey on the previous work done on fault tolerant learning. Those analytical works that have been reported in the literature and those algorithms that have been proposed to deal with weight noise or node fault will be elucidated. Furthermore, an objective function based framework for fault tolerant learning is proposed. Future work along the direction is presented.

Keywords : Gradient Descent, Fault Tolerance, KL Divergence, Learning Theory, Neural Networks

1 Introduction

In conventional learning theory, the objective of a learning algorithm is to attain a neural network (NN) of good generalization. To accomplish this, one approach is by adding regularizer [32, 31, 37, 38] to penalize the weights' magnitude. Another approach is by pruning [23, 29, 33, 31, 48, 43] redundant weights. The purposes of weight penalization and redundant weights removal are essentially the same, i.e. to reduce model complexity. From statistical learning theory, [1] over-complexity can always lead to poor generalization (over-fit). In other words, conventional learning theory aims at looking for a neural network that is of minimal complexity. This idea works very well whenever a trained neural network model is hard-coded in a program running in a computer. The floating point format in a computer can represent a weight value of very high precision. The performance of this software implemented model will be indifference from the one obtained after learning. However, it is not true for a neural network that is embedded in a hardware system, like FPGA.

As mentioned in [44], hardware can fail in a number of complex ways. Stuck at fault affecting the sign bit of a weight might cause its value to change from $+W$ to $-W$ or vice versa. An open circuit in an analog VLSI implementation leading to a disconnection of a link could be modeled by setting its weight stuck at zero or a neuron dead. If a link got shorted to power supply, the fault could be abstracted by setting the associated weight stuck at $+W$ or $-W$. If a weight is implemented as resistors, its value may degrade over time due to material depreciation. Nevertheless, the reduced precision floating point representation will lead to quantization error

[1] Please refer to Chapter 9 in [8] and Chapter 7 in [24]

which can be modeled as a multiplicative weight noise effect. In summary, four types of fault that can found in a neural network.

- Node fault (Stuck-at fault) – Single node fault or multiple nodes fault
- Weight noise (weight perturbation) – Additive or multiplicative
- Input noise (input perturbation)– Additive or multiplicative
- Perturbations on model parameters – e.g. the centers and widths in RBF, and the parameter τ in the sigmoid function
- Single event upset – random output/input upset of an electronic component due to radiation

All these hardware failures and low precision floating point representation will eventually make the performance of an FPGA implemented neural network degrade from its software counterpart. This degradation can be drastic if their effects have not been properly considered during training.

Tremendous research works have been done in the last decade to deal with these network faults. Basically, those works can be categorized into one of the following areas.

- Analytical works, which include (I) the analysis on effect of those faults on (1) the output sensitivity, (2) the error sensitivity and (3) the probability of output error of a neural network; and (II) the analysis on the relationship amongst fault tolerance, generalization and model complexity.
- Develop fault tolerant learning algorithms or network synthesis methods dealing with weight noise.
- Develop fault tolerant learning algorithms or network synthesis methods dealing with node fault.

The remind of the paper is organized as follows. In the next section, previous work on sensitivity analysis on a neural network suffered from noise and fault will be presented. Section 3 presents the algorithms and synthesis methods that can train a neural networks to deal with weight noise. Then, those algorithms and synthesis methods for dealing with node fault will be elucidated in Section 4. An objective function based framework will be presented in Section 5. Disucssion on the previous works done and a list of suggested future works will be given in Section 7. The conclusion is presented in Section 8.

2 Analytical work

Consider a Madaline with threshold logic output neuron, Stevenson *et al* [54] gave a comprehensive analysis on the *probability of output error* with respect to different type of noises, such as input and weight noise, both additive and multiplicative. Choi and Choi [15] based on statistical sensitivity approach deriving different *output sensitivity measures* of a MLP network due to different type of noise. For a Madaline with sigmoidal output neuron, Piche in [47] followed an idea from signal to noise ratio (SNR) and derived a set of measures for the output sensitivity. Furthermore, a weight accuracy selection algorithm is developed and applied to determine the precision requirement in hardware implementation. Townsend and Tarassenko [65] studied the radial basis function (RBF) network with multiple outputs and derived a matrix form output sensitivity for an RBF that is suffered from perturbations in input data, basis centers and output weights.

Table 1: Research works on the analysis of a fault tolerant NN.

Year/Ref.	Fault	Model	Work
1990 [54]	Any weight noise	Madaline	Probability of output error
1992 [15]	Any noise	Any	Output sensitivity measure
1995 [47]	Any noise	Madaline	Precision requirement
1995 [44]	Single node fault	MLP with 0/1 output	Triple modular redundant
1996 [10]	Multiplicative weight noise	RBF	Generalization ability
1999 [65]	Any noise	RBF	Output sensitivity matrix
1999 [2]	Any weight noise	MLP	Output sensitivity measure
1999 [45]	-	MLP	Relationship between FT, generalization and VC dim.
2000 [4]	Any weight noise	MLP	Generalization ability
2003 [56]	Single node fault	RBF	Error sensitivity measure
2004 [20]	Any weight noise	Functional net	Error sensitivity measure
2004 [13]	Any noise	MLP	Boundedness on the MSE wrt input/weight noise
2005 [17, 18]	Any noise	RBF	Upper bounds on the MSE wrt inputs/parameters noise
2005 [57]	Multiple nodes fault	RBF	Fault tolerant regularizer
2007 [55]	Multiplicative weight noise	RBF	Equivalence between explicit regularization and weight decay

Any noise here refers to the noise which is either input noise or weight noise, and either additive or multiplicative.

As output sensitivity is an indirect view point to understand the effect of NN due to noise, the actual performance degradation cannot be identified easily. An alternative approach is to analyze the actual performance, i.e. generalization, being affected. Catala and Parra proposed a fault tolerance parameter model and studied the performance degradation of a RBF network if its basis centers, widths and the 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] and RBF network [6]. Similarly, Fontenla-Romero *et al* derived the *error sensitivity measures* for functional net [20] Sum & Leung [56] derived an *error sensitivity measure* for single node fault RBF network. Then an on-line pruning algorithm making use of the measure is proposed to attain fault tolerant RBF.

Noise can sometimes be beneficial to neural network as well. Murray & Edwards [40] investigated and found that adding multiplicative weight noise (and other kinds of noise) during training can improve the generalization ability of a MLP. Bishop [7] showed that the effect of adding small additive input noise during training is equivalent to Tikhonov regularization. Jim *et al* [27] 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.

3 Methods Dealing With MWN

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. Hammadi and Ito [22] proposed to shrink the magnitude of the most harmful weight, that is defined by a relevance measure, during each step of training. Cavalieri & Mirabella in [11] have proposed a modified backpropagation learning algorithm for multilayer perceptrons. 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. Kamiura *et al* [28] proposed another weight limiting step. In which the feasible range is decided from the average and variance of the weights' magnitude.

Consider that the noise effect can eventually be cancelled out at the output node if all the weight values are equal, Simon in [53] suggested a distributed fault tolerance learning approach for optimal interpolation net. The learning is formulated 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 [42] demonstrated how a fault tolerant RBF network can be obtained by using a weight decay regularizer [37]. 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.

4 Methods Dealing With Node Fault

To overcome the effect due to node fault, two approaches have been adopted : (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 [50, 9] or together with random node deletion and addition [14] during training have been developed. Adding network redundancy by replicating multiple hidden layers after a NN has been well trained [19, 44] is another one. Under the same scenario, limit weight magnitude either by adding weight decay regularizer [14] or bounding the weight magnitude to a small value during training [11, 22, 28] are another two techniques that can succeed in obtaining a fault tolerant NN. Sher & Hsieh [51] proposed a constraint backpropagation algorithm training method. In which, a weight will be updated if any one of the faulty networks (with m nodes fault) gives absolute error larger than a threshold. The weight is thus updated by applying backpropagation algorithm on the faulty network which gives the largest absolute error[2].

Another approach is to formulate the learning directly as a constraint optimization problem. Neti et al [41] 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 [52] and Phatak & Tcherner in [46] formulated the learning problem to a simpler unconstraint

[2]The measure being used in Sher & Hsieh's work is basically the same as the one being used in Deodhare *et al*'s work [16].

Table 2: Training algorithms and network synthesis methods.

Year/Ref.	Fault	Model	Idea
1991 [9, 50]	Node fault	MLP	Injecting random node fault during training
1992 [41]	Single node fault	MLP	Minimax Problem[1]
1993 [19, 44]	Single node fault	MLP	Adding redundancy
1993 [39]	Weight noise	MLP	Adding weight noise during training
1994 [14]	Weight noise	MLP	Apply weight decay algo.[2]
1997 [22]	Node fault	MLP	Weight magnitude bounding
1998 [16]	Single node fault	MLP	Minimax problem[3]
1999 [11]	Node fault	MLP	Weight magnitude bounding
1999 [51]	Multiple nodes fault	MLP	Constraint BP[3]
2000 [28]	Node fault	MLP	Weight magnitude bounding
2000 [42]	Multiplicative weight noise	RBF	Apply weight decay algo.
2000 [3, 5]	Multiplicative weight noise	MLP	Explicit regularization
2001 [53]	Weight noise	IN[a]	Nonlinear program[4]
2002 [46]	Single node fault	MLP	Nonlinear program[5]
2004 [62]	Node fault	MLP	Apply penalty term[6]
2005 [34, 57]	Multiple nodes fault	RBF	Fault tolerant regularizer
2007 [61]	Multiplicative weight noise	RBF	Apply KL divergence

[a] Interpolation net

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

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

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

[4] Minimizing mean square errors subject to equality constraint on weights' magnitudes

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

[6] The objective function is $1/N \sum_{k=1}^{N}(y_k - f(x_k, \theta))^2 + \frac{\alpha}{n}|\theta|^n$

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 difficulty in extending the existing approaches to multiple nodes fault, Leung *et al* in [34] and Sum in [57] have attempted to these problems by devising a new regularizer for fault tolerant learning.

5 Objective Function Based Framework

5.1 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

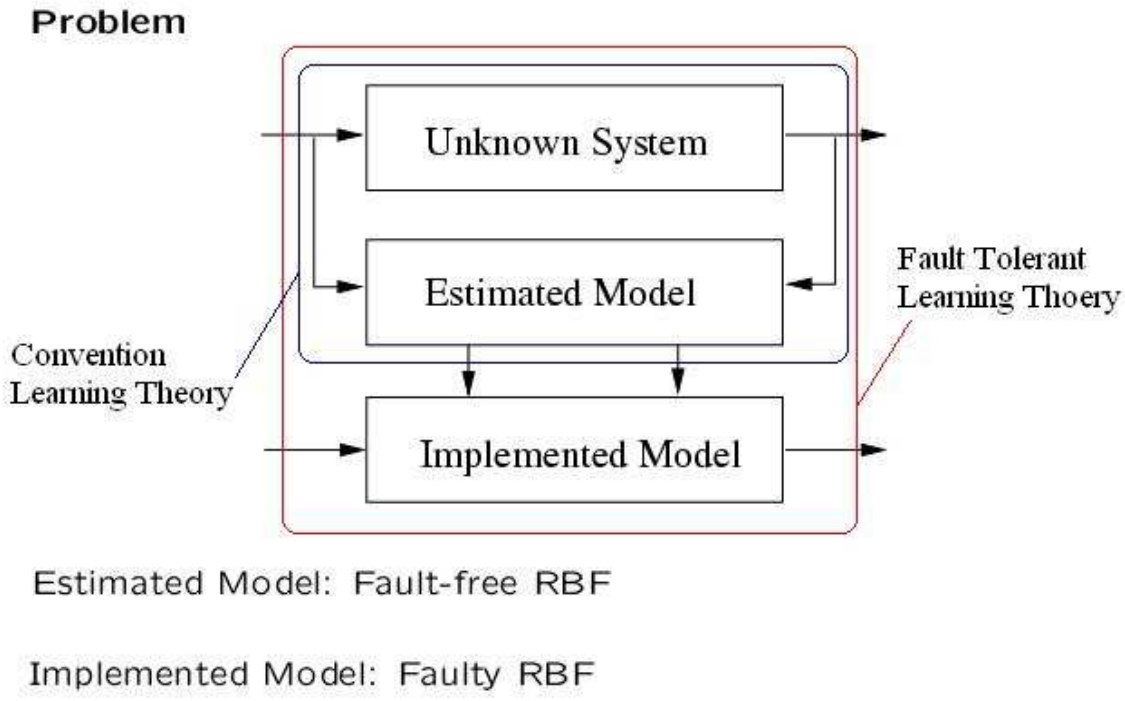

Figure 1: Conventional learning theory and fault tolerant learning theory.

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 $P(\tilde{\mathcal{M}}|\hat{\mathcal{M}})$. This is depended on the property of the fault model concerned. It could be very complicated if multiple fault models co-exist.

5.2 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 $\mathcal{M}_0$.

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

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

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.

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

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

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

5.3 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

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

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

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

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

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.

$$\mathcal{M} \; : \quad y_k = \sum_{i=1}^{M} \theta_i \phi_i(x_k) + e_k \tag{6}$$

$$e_k \sim \mathcal{N}(0, S_e), \tag{7}$$

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 :

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

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 :

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

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 regularrization term $R(\theta)$ in Equation (10) has the following properties :

$$\begin{aligned} (i)\ \ R(\theta) &\geq 0, \text{ for all } \|\theta\| \geq 0 \\ (ii)\ \ R(0) &= 0 \text{ and} \\ (iii)\ \ R(\theta') &> R(\theta), \text{ if } \|\theta'\| > \|\theta\|. \end{aligned}$$

Two examples for $R(\theta)$ are $\sum_{i=1}^{M}\theta_i^2$, which is from weight decay [14] and $\sum_{i=1}^{M}\|\theta_i\|^n$ (n is a positive integer) which is from [62]. 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 :

$$\mathcal{L}(\theta|\mathcal{D}) = \int_{\tilde{\theta}\in\tilde{\Omega}_\theta} J(\tilde{\theta}|\mathcal{D})P(\tilde{\theta}|\theta)d\tilde{\theta} \tag{15}$$

$$\hat{\theta}_I = \arg\min_{\theta\in R^M}\left\{\mathcal{L}(\theta|\mathcal{D})\right\}. \tag{16}$$

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

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

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

5.4 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.

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

Multiplicative weight noise exists whenever a weight is encoded in a low precision binary form. An implementation of a model θ (denoted by $\tilde{\theta}$) can be defined as follows :

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

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

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

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

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

So,

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

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.

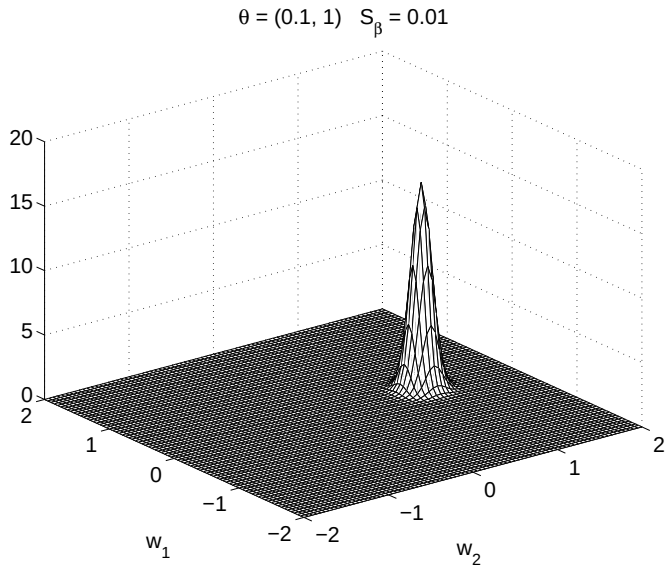

Figure 2: For multiplicative weight noise case, the conditional probability $P(\tilde{\theta}|\theta)$ for θ equals to $(0.1, 1)^T$.

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,

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

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

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

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

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

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

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

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 :

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

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.

5.4.2 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

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

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 $k = 1, 2, \cdots, N$. $S(x,\theta) = S_e + S_\beta\sum_{i=1}^M \phi_i^2(x)\theta_i^2$. The likelihood probability will be given as follows :

$$P(\mathcal{D}|\theta) = \prod_{k=1}^N \int P(y_k|x_k,\beta,\theta)P(\beta)d\beta. \quad (32)$$

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

$$\begin{aligned} \mathcal{L}(\theta|\mathcal{D}) &= -\sum_{k=1}^N \log\int P(y_k|x_k,\beta,\theta)P(\beta)d\beta \quad (33) \\ &= \frac{1}{2}\log 2\pi + \frac{1}{2N}\sum_{k=1}^N \log S(x_k,\theta) + \frac{1}{2N}\sum_{k=1}^N \frac{(y_k-\phi^T(x_k)\theta)^2}{S(x_k,\theta)}. \quad (34) \end{aligned}$$

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

$$\arg\min_\theta\left\{\frac{1}{2N}\sum_{k=1}^N \log S(x_k,\theta) + \frac{1}{2N}\sum_{k=1}^N \frac{(y_k-\phi^T(x_k)\theta)^2}{S(x_k,\theta)}\right\}. \quad (35)$$

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 (34), it is readily obtained

$$\begin{aligned} \frac{\partial}{\partial\theta}\log S(x_k,\theta) &= \frac{2S_\beta}{S(x_k,\theta)}\mathbf{G}(x_k)\theta, \quad (36) \\ \frac{\partial}{\partial\theta}\frac{(y_k-\phi^T(x_k)\theta)^2}{S(x_k,\theta)} &= -\frac{2S_\beta(y_k-\phi^T(x_k)\theta)^2}{S^2(x_k,\theta)}\mathbf{G}(x_k)\theta - \frac{2(y_k-\phi^T(x_k)\theta)}{S(x_k,\theta)}\phi(x_k), \quad (37) \end{aligned}$$

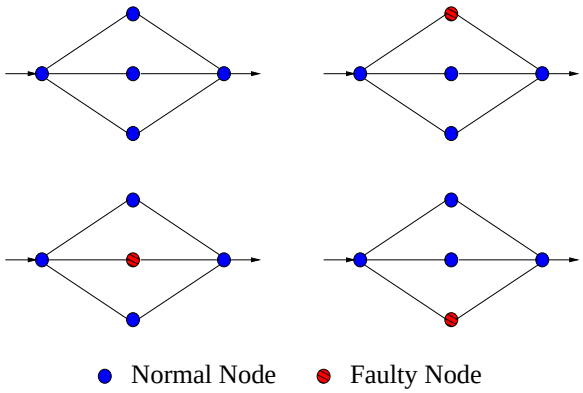

Figure 3: Single-node fault NN models. For a network of M hidden nodes, there are M possible single-node fault models.

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 :

$$\theta(t+1) = \theta(t) - \mu \frac{\partial}{\partial \theta} \mathcal{L}(\theta(t)|\mathcal{D}), \tag{38}$$

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

$$\frac{\partial \mathcal{L}(\theta|\mathcal{D})}{\partial \theta} = \frac{S_\beta}{N} \sum_{k=1}^{N} \left(\frac{1}{S(x_k,\theta)} - \frac{(y_k - \phi^T(x_k)\theta)^2}{S^2(x_k,\theta)} \right) \mathbf{G}(x_k)\theta - \frac{1}{N} \sum_{k=1}^{N} \frac{(y_k - \phi^T(x_k)\theta)}{S(x_k,\theta)} \phi(x_k). \tag{39}$$

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

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

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,

$$\Omega = R^M, \quad \tilde{\Omega}_\theta = \{\theta, \theta_{-1}, \theta_{-2}, \cdots, \theta_{-M}\}. \tag{40}$$

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

$$P(\tilde{\theta}|\theta) = \begin{cases} 1-q & \text{if } \tilde{\theta} = \theta \\ q/M & \text{if } \tilde{\theta} = \theta_{-1} \\ \vdots & \vdots \\ q/M & \text{if } \tilde{\theta} = \theta_{-M}. \end{cases} \tag{41}$$

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

$$\mathcal{L}(\theta|\mathcal{D}) = (1-q)J(\theta|\mathcal{D}) + \frac{q}{M} \sum_{i=1}^{M} J(\theta_{-i}|\mathcal{D}). \tag{42}$$

In which,

$$J(\theta_{-i}|\mathcal{D}) \;=\; J(\theta|\mathcal{D}) + \theta_i^2 g_i + \; 2\theta_i \frac{1}{N}\sum_{k=1}^{N}(y_k - \phi^T(x_k)\theta)\phi_i(x_k) \tag{43}$$

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 :

$$\mathcal{L}(\theta|\mathcal{D}) \;=\; J(\theta|\mathcal{D}) + \frac{2q}{M}\frac{1}{N}\sum_{k=1}^{N} y_k\phi^T(x_k)\theta + \; \frac{q}{M}\theta^T[Q_g - 2H_\phi]\theta. \tag{44}$$

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

$$\hat{\theta}_I \;=\; \left(H_\phi + \frac{q/M}{1-q/M}Q_g\right)^{-1}\frac{1}{N}\sum_{k=1}^{N} y_k\phi(x_k). \tag{45}$$

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

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

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

$$\tilde{\theta}_i = \beta_i\theta_i, \tag{46}$$

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

$$f(x,\theta,\beta) = \sum_{i=1}^{M}\beta_i\theta_i\phi_i(x). \tag{47}$$

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.

$$P(\beta_i) = \begin{cases} p & \text{if } \beta_i = 0 \\ 1-p & \text{if } \beta_i = 1. \end{cases} \tag{48}$$

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

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

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

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 :

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

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

$$\mathcal{L}(\theta|\mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - \phi^T(x_k)\theta\right)^2 + \theta^T\Sigma\theta, \tag{50}$$

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 $\theta^T\Sigma\theta$.

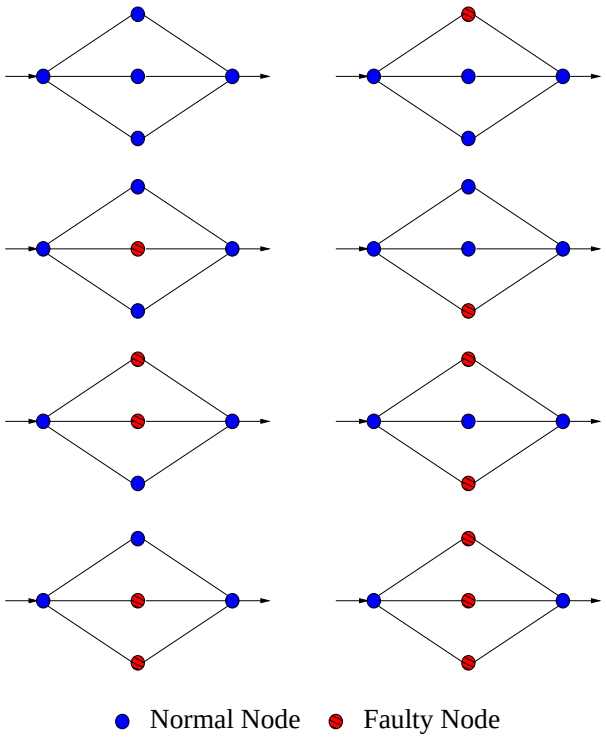

Figure 4: Multiple-nodes fault NN models. For a network of n hidden nodes, there are $2^n - 1$ possible multiple-nodes fault models.

6 Discussion & Future Works

As most of earlier works were working on training algorithms for fault tolerant neural network and demonstrated their success via intensive computer simulations. Only a few recent papers have researched on the relationship amongst fault tolerant, generalization and model complexity [45, 57, 58]. A complete picture between conventional learning theory and fault tolerant learning is still unclear. Objective function based framework is in its initial stage and a lot more works have to be done to fill the blanks within. Many questions are left to be answered.

- 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 their objective functions are found, what are their similarities, differences and relationships with those defined in conventional learning ?
- Some research articles in the literature have summarized the previous works in regard to fault tolerant neural networks. But, little theoretical work has been done and almost no previous work has been done along the statistical learning point of view.

A summary of the known objective functions for fault tolerant learning is depicted in Table 3. In which, the corresponding measures for Chiu *et al*'s work [14] and Deodhare *et al*' work [16] are

Table 3: Summary of the objective functions for FT learning.

Year	$\mathcal{L}(\theta\|\mathcal{D})$	Objective Function
1992 [41]	$\min_\theta\{\max_{\tilde{\theta}} \mathrm{SSE}(\tilde{\theta})\}$	$\min_\theta\{\max_{\tilde{\theta}} \sum_{k=1}^N (y_k - f(x_k, \tilde{\theta}\|\theta))^2\}$
1994 [14]	Unknown	$\mathrm{SSE}(\theta) + \lambda\theta^T\theta, \ \ (\lambda > 0)$
1998 [16]	Unknown	$\min_\theta\{\max_{\tilde{\theta}} \max_k (y_k - f(x_k, \tilde{\theta}\|\theta))^2\}$
2000 [3, 55]	$\int_{\tilde{\theta}} \mathrm{SSE}(\tilde{\theta}) P(\tilde{\theta}\|\theta) d\tilde{\theta}$	$\mathrm{SSE}(\theta) + S_\beta \theta^T \left[\frac{1}{N}\sum_{k=1}^N \mathbf{G}(x_k)\right] \theta$
2002[a] [46]	$\sum_{\tilde{\theta}} \mathrm{SSE}(\tilde{\theta}) P(\tilde{\theta}\|\theta)$	$\mathrm{SSE}(\theta) + \frac{\alpha}{\|\Omega_{\tilde{\theta}}\|N} \sum_{\tilde{\theta}\in\Omega_{\tilde{\theta}}} \sum_{k=1}^N (y_k - f(x_k, \tilde{\theta}\|\theta))^2$
2004 [62]	Unknown	$\mathrm{SSE}(\theta) + \lambda \sum_{i=1}^M \|\theta_i\|^n, \ \ (\lambda > 0)$
2005 [34, 57]	$\sum_{\tilde{\theta}} \mathrm{SSE}(\tilde{\theta}) P(\tilde{\theta}\|\theta)$	$\mathrm{SSE}(\theta) + p\ \theta^T (Q_g - H_\phi)\theta$
2007 [61]	$-\int_{\tilde{\theta}} \log P(\mathcal{D}\|\tilde{\theta}) P(\tilde{\theta}\|\theta) d\tilde{\theta}$	$\sum_{k=1}^N \log S(x_k, \theta) + \sum_{k=1}^N \frac{(y_k - \phi^T(x_k)\theta)^2}{S(x_k,\theta)}$
2007 [60]	$\sum_{\tilde{\theta}} \mathrm{SSE}(\tilde{\theta}) P(\tilde{\theta}\|\theta)$	$\mathrm{SSE}(\theta) + \frac{2q}{MN} \sum_{k=1}^N y_k \phi^T(x_k)\theta + \frac{q}{M}\theta^T[Q_g - 2H_\phi]\theta$

$\mathrm{SSE}(\theta) = N^{-1} \sum_{k=1}^N (y_k - f(x_k, \theta)^2 \qquad S(x, \theta) = S_e + S_\beta \sum_{i=1}^M \phi_i^2(x)\theta_i^2$

[a] $P(\theta|\theta) = \frac{1}{1+\alpha}$, $P(\tilde{\theta}|\theta) = \frac{\alpha}{|\Omega_{\tilde{\theta}}|(1+\alpha)}$

still unknown. In addition to the literature survey provided in the earlier sections, some specific problems that are still open are listed below.

- What is the underlying objective function for injecting random node fault during training ?
- What is the underlying objective function for adding multiplicative weight noise during training ?
- What is the corresponding measure $\mathcal{L}(\theta|\mathcal{D})$ for the objective function $\mathrm{SSE}(\theta) + \lambda\theta^T\theta$?
- What is the corresponding measure $\mathcal{L}(\theta|\mathcal{D})$ for the objective function $\min_\theta\{\max_{\tilde{\theta}} \max_k (y_k - f(x_k, \tilde{\theta}|\theta))^2\}$?
- What is the performance of a neural network if it is trained by $-\int_{\tilde{\theta}} P(\mathcal{D}|\tilde{\theta}) P(\tilde{\theta}|\theta) d\tilde{\theta}$, $-\left\{\int_{\tilde{\theta}} P(\tilde{\theta}|\theta) \log P(\mathcal{D}|\tilde{\theta}) d\tilde{\theta}\right\}$ or $-\log\left\{\int_{\tilde{\theta}} P(\mathcal{D}|\tilde{\theta}) P(\tilde{\theta}|\theta) d\tilde{\theta}\right\} - \log P(\theta)$?

7 Conclusions

Fault tolerant learning is essentially a difficult but interesting topic deserved for further investigation. This paper, extended from our previous work in [58], has given a comprehensive survey on the related work done in the past two decades. Besides, an objective function based framework has been proposed as a theoretical foundation for the undertanding of fault tolerant learning. A number of open problems have been listed. In which, training algorithms using KL-divergence and *a posterior* probability should be deserved for special attention.

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] Ralf Erickhoff and Ulrich Ruckert, Robustness of radial basis functions. *Neurocomputing*, Vol.70, Issue 16-18, pp.2758-2767, October 2007.

[19] 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.

[20] Fontenla-Romero O. *et al*, A measure of fault tolerance for functional networks, *Neurocomputing*, Vol.62, 327-347, 2004.

[21] Ali Ghodsi and Dale Schuurmans, Automatic basis selection techniques for RBF networks. *Neural Networks*, Vol-16, No. 5-6, pp.809-816, 2003.

[22] Nait Charif Hammadi and Hideo Ito, A learning algorithm for fault tolerant feedforward neural networks, *IEICE Transactions on Information & Systems*, Vol. E80-D, No.1, 1997.

[23] Hassibi B and D.G. Stork, Second order derivatives for network pruning: Optimal brain surgeon. In Hanson *et al.* (eds) *Advances in Neural Information Processing Systems*, 164-171, 1993.

[24] T. Hastie, R. Tibshirani, J. H. Friedman *The Elements of Statistical Learning*, Springer, 2001.

[25] S. Haykin, *Neural networks: A comprehensive foundation*, Macmillan College Publishing Company, Inc. 1994.

[26] Holt J. and J. Hwang, Finite precision error analysis of neural networks hardware implementation, *IEEE Transactions on Computers*, Vol.42, p.281-290, 1993.

[27] 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.

[28] Naotake Kamiura *et al*, On a weight limit approach for enhancing fault tolerance of feedforward neural networks, *IEICE Transactions on Information & Systems*, Vol. E83-D, No.11, 2000.

[29] LeCun Y. *et al.*, Optimal brain damage, *Advances in Neural Information Processing Systems 2* (D.S. Touretsky, ed.) 396-404.

[30] Kullback S., *Information Theory and Statistics*, Wiley, 1959.

[31] 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.

[32] 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.

[33] 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.

[34] Chi-sing Leung and John Sum, Fault tolerant regularizer for multiple nodes fault RBF, *IEEE Transactions on Neural Networks*, Vol. 19 (3), pp.493-507, 2008.

[35] Mackay D.J.C., A Practical Bayesian Framework for Backprop Networks, *Neural Computation*, Vol.4(3) 448-472.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] Parra X. and A. Catala, Fault tolerance in the learning algorithm of radial basis function networks, *Proc. IJCNN 2000*, Vol.3, 527-532, 2000.

[43] 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.

[44] 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.

[45] Phatak D.S., Relationship between fault tolerance, generalization and the Vapnik-Cervonenkis (VC) dimension of feedforward ANNs, *IJCNN'99*, Vol.1, 705-709, 1999.

[46] Phatak D.S. and E. Tcherner, Synthesis of fault tolerance neural networks, *Proc. IJCNN'02*, 1475-1480, 2002.

[47] Piche S.W., The selection of weight accuracies for Madalines, *IEEE Transactions on Neural Networks*, Vol.6, 432-445, 1995.

[48] Reed R., Pruning algorithms – A survey, *IEEE Transactions on Neural Networks*, Vol.4(5), 740-747, 1993.

[49] R. Reed, R.J. Marks II & S. Oh, Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter, *IEEE Transactions on Neural Networks*, Vol.6(3), 529-538, 1995.

[50] Sequin C.H. and R.D. Clay, Fault tolerance in feedforward artificial neural networks, *Neural Networks*, Vol.4, 111-141, 1991.

[51] Buh Yun Sher and Weng-Shong Hsieh, Fault Tolerance Training of Feedforward Neural Networks. *Proceedings of the National Science Council, Republic of China (A),* Vol-23, No.5, pp. 599-608, 1999.

[52] Simon D. and H. El-Sherief, Fault-tolerance training for optimal interpolative nets, *IEEE Transactions on Neural Networks*, Vol.6, 1531-1535, 1995.

[53] Simon D., Distributed fault tolerance in optimal interpolative nets, *IEEE Transactions on Neural Networks*, Vol.12(6), 1348-1357, 2001.

[54] 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.

[55] 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, accepted for publication in *IEEE Transactions on Neural Networks*.

[56] John Sum and Chi-sing Leung, On the error sensitivity measure for pruning RBF networks, *Proceedings of ICMLC 2003, Xi'an China*, pp.1161-1167, 2003.

[57] 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.

[58] John Sum, Towards an objective function based framework for fault tolerant learning, in *Proceedings of TAAI'2007*.

[59] John Sum and Chi-sing Leung, Prediction error of a fault tolerant neural network, in submission to *Neurocomputing*.

[60] John Sum, Chi-sing Leung, Lipin Hsu, Yu-fa Huang, An objective function for single node fault RBF learning, in *Proceedings of TAAI'2007*.

[61] John Sum, Chi-sing Leung and Lipin Hsu, Fault tolerant learning using Kullback-Leibler Divergence, in *Proc. TENCON'2007* Taipei, 2007.

[62] H. Takase, H. Kita and T. Hayashi, A study on the simple penalty term to the error function from the viewpoint of fault tolearnt training. 2004.

[63] Elko B. Tchernev, Rory G. Mulvaney, and Dhananjay S. Phatak, Investigating the Fault Tolerance of Neural Networks, *Neural Computation*, Vol.17, 1646-1664, 2005.

[64] 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.

[65] 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.

[66] 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.

[67] Jihan Zhu and Peter Sutton, FPGA implementation of neural networks – a survey of a decade of progress, *Proceedings of the 13th International Conference on Field Programmable Logic and Applications*, Lisbon, 2003.