Convergence Analysis of Node Fault Injection During Training

Kevin Ho
Dept. of Computer Science and Communication Engineering
Providence University
Sha-Lu, Taiwan
Email: ho@pu.edu.tw

Chi-sing Leung
Department of Electronic Engineering
City University of Hong Kong
Kowloon Tong, Hong Kong
Email: eeleungc@cityu.edu.hk

John Sum, Siu-chung Lau
Institute of Technology Management
National Chung Hsing University
Taichung, Taiwan.
Email: pfsum@nchu.edu.tw

***Abstract*—Improving fault tolerance of a neural network is an important issue that has been studied for more than two decades. Various algorithms have been proposed in sequel and many of them have succeeded in attaining a fault tolerant neural network. Amongst all, on-line node fault injection-based algorithms are one type of these algorithms. Despite its simple implementation, theoretical analyses on these algorithms are far from complete. In this paper, an on-line node fault injection training algorithm is studied. By node fault injection training, we assume that the hidden nodes are random neuron in which the output of these hidden nodes can be zeros in a random manner. So, in each step of update, we randomly set the hidden outputs to be zeros. The network output and the gradient vector are calculated with these zero-output hidden nodes, and thus apply the standard online weight algorithm to update the weight vector. The corresponding objective function is derived and the convergence of the algorithm is proved. By a theorem from H. White, we show that the weight vector obtained by this algorithm can converge with probability one. The weight vector converges to a local minimum of the objective function derived.**

***Keywords*-convergence; learning; MLP; node fault;**

I. Introduction

Regularization [26], [25], [31], [32], [41] and pruning [16], [24], [27], [25], [38], [36] are common techniques to attain a neural network with good generalization. These techniques work well under the assumption that the trained neural networks can be ideally implemented (i.e. fault-free implementation). However, in electronic implementations (like FPGA [17]), component failure, sign bit change, open circuit [37], finite precision [39] and exposure to radiation [48] will exist. If special care is not considered, the performance of a neural network could degrade drastically.

Thus, various methods have been proposed to attain a neural network that is able to tolerate fault/noise. Some of these methods include injecting random node fault during training [6], [40], applying weight decay learning [7], [10], [29], injecting weight noise during training [12], [33], [34], introducing network redundancy [37], formulating the training as a minmax problem [11], [35], hard-bounding weight magnitude during training [8], [15], [23], regularization, [2], [3], [4], [28], [43]. and others [42].

Amongst all, injecting fault/noise during training is almost the simplest and effective method [**?**], [34], [31], [**?**], [46], [51]. Sequin & Clay [40] and Bolt [6] are the first two groups proposing injecting random node fault during training, and showing by simulations that the resultant multilayer perceptron (MLP) is able to tolerate random node fault. Instead of injecting node fault, Judd & Munro [22] proposed training a MLP with random output hidden nodes. On the other hand, Murray & Edward [33], [34] proposed injecting weight noise during training. By simulations, they showed that the resultant MLP is able to tolerate weight fault and weight noise. Besides, the convergence of this training method is better than the standard back-propagation. Jim *et al* [21] applied the same idea to real-time-recurrent-learning (RTRL). Similarly, they showed that the convergence of this modified RTRL is better than the standard RTRL. Moreover, the resultant RNN has better generalization than the one obtained by the standard RTRL. Apart from injecting fault/noise during training, injecting input noise during training is another simple method [5], [30], [20]. Its idea is similar. Random noise is injected to the input of a network during training.

Although many on-line fault/noise injection learning algorithms have been developed in the last two decades, not many theoretical works have been done to analyze their success/failure. Most of them are focused on *the effect of noise/fault on the network output or the prediction error of a neural network*. Analyses on the convergence and the objective functions of these training algorithms are scarce [1], [13], [14], [45].

Therefore, we have recently investigated the convergence and the objective functions of some on-line fault/noise injection training algorithms applying for radial basis function (RBF) networks [18], [19], [44]. For injecting weight noise during training, we showed that its convergence is with probability one and the corresponding objective function is mean square errors (MSE). It implies that this algorithm is not able to improve generalization, nor fault tolerance. For injecting node fault during training, we showed that its convergence is with probability one and the corresponding objective function is identical to the objective function of a regularization algorithm in which a fault tolerant regularizer is added [28].

While some breakthrough analyses have been done for RBF, the convergence properties and the objective functions of applying these algorithms for MLPs are still unknown.

Thus, we apply a theorem from H.White [49] and follow the approach in [50] to prove the convergence of this online node fault injection-based training algorithm for MLPs and identify the objective function for this algorithm.

The rest of the paper will be devoted to this analysis. In the next section, the model of a MLP with single linear output node will be defined. Then, the node fault injection-based training algorithm will be described in Section III. The main results are presented in Section IV and Section V. The mean update equation of training algorithm will be presented in Section IV, and the objective function of the algorithm based on this mean update equation will be derived. In Section V, the convergence proof of the algorithm is presented. Finally, Section VI gives the conclusion.

II. Background

We assume that the training data set $\mathcal{D} = \{(\mathbf{x}_k, y_k)\}_{k=1}^N$ is generated by an unknown system, where $\mathbf{x}_k \in R^n$ is the input and $y_k \in R$ is the output.

A. Network Model

This unknown system is thus approximated by a MLP with n input nodes, m hidden nodes, and one linear output node, defined as follows :

$$f(\mathbf{x}_k, \mathbf{d}, \mathbf{A}, \mathbf{c}) = \mathbf{d}^T \mathbf{z}(\mathbf{A}^T \mathbf{x}_k + \mathbf{c}), \tag{1}$$

where $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_m] \in R^{n \times m}$ is the input-to-hidden weight matrix, $\mathbf{a}_i \in R^n$ is the input weight vector associated with the i^{th} hidden node, $\mathbf{c} = (c_1, \cdots, c_m)^T \in R^m$ is the input-to-hidden bias vector, $\mathbf{d} \in R^m$ is the hidden-to-output weight vector, and $\mathbf{z} = (z_1, \cdots, z_m)^T \in R^m$ is output vector of the hidden layer. It is a vector function in which the i^{th} element is a function of input $\mathbf{x}_k$, the input weight vector $\mathbf{a})i$ and the input bias c_i.

$$z_i(\mathbf{x}_k, \mathbf{a}_i, c_i) = \frac{1}{1 + \exp(-(\mathbf{a}_i^T \mathbf{x}_k + c_i))} \tag{2}$$

for $i = 1, 2, \cdots, m$.

For the sake of presentation, we let $\mathbf{w}_i$ be the parametric vector associated to the i^{th} hidden node, where

$$\mathbf{w}_i = (d_i, \mathbf{a}_i^T, c_i)^T. \tag{3}$$

Besides, we let $\mathbf{w}$ be a parametric vector augmenting all the parametric vectors $\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_m$, i.e.

$$\mathbf{w} = (\mathbf{w}_1^T, \mathbf{w}_2^T, \cdots, \mathbf{w}_m^T)^T. \tag{4}$$

In other word, the output of the network can be denoted as $f(\mathbf{x}_k, \mathbf{w})$. Throughout the paper, we call $\mathbf{w}_1$, $\mathbf{w}_2, \cdots, \mathbf{w}_m$ and $\mathbf{w}$ the **weight vectors**.

B. Weight Decay Training Algorithm

In weight decay training, a sample is randomly drawn from the dataset $\mathcal{D}$ at each update step. We denote the sample being selected at the t^{th} step as $\{\mathbf{x}_t, y_t\}$. Formally speaking, we can let $\delta_1(t), \delta_2(t), \cdots, \delta_N(t)$ be N independent binary random numbers and $\sum_{k=1}^N \delta_k(t) = 1$ for all $t \geq 0$. For $t_1 \neq t_2$, random vectors $(\delta_1(t_1), \cdots, \delta_N(t_1))^T$ and $(\delta_1(t_2), \cdots, \delta_N(t_2))^T$ are independent. Then, $(\mathbf{x}_t^T, y_t)^T$ can formally be defined as the follows :

$$\begin{bmatrix} \mathbf{x}_t \\ y_t \end{bmatrix} = \sum_{k=1}^N \delta_k(t) \begin{bmatrix} \mathbf{x}_k \\ y_k \end{bmatrix}. \tag{5}$$

The input $\mathbf{x}_t$ is thus fed in the MLP, and the output is calculated by (1) and (2). To shorten the length of the equations, we denote $z_i(\mathbf{x}_t, \mathbf{a}_i(t), c_i(t))$ in (2) by $z_i(t)$ in the rest of the paper. The update equations of the weight vectors $\mathbf{w}_i$ (for $i = 1, 2, \cdots, m$) can then be written as follows :

$$\begin{aligned} & \mathbf{w}_i(t+1) - \mathbf{w}_i(t) \\ = \; & \mu(t) \left\{ (y_t - f(\mathbf{x}_t, \mathbf{w}(t))) \mathbf{g}_i(\mathbf{x}_t, \mathbf{w}(t)) \right. \\ & \left. - \; \alpha \mathbf{w}_i(t) \right\}, \end{aligned} \tag{6}$$

where

$$\mathbf{g}_i(\mathbf{x}_t, \mathbf{w}(t)) = \left. \frac{\partial}{\partial \mathbf{w}_i} f(\mathbf{x}_t, \mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}(t)} \tag{7}$$

$$= \begin{bmatrix} z_i(t) \\ d_i(t) z_i(t)(1 - z_i(t)) \mathbf{x}_t \\ d_i(t) z_i(t)(1 - z_i(t)) \end{bmatrix}, \tag{8}$$

$\mu(t) > 0$ is the step size at the t^{th} step, and $\alpha > 0$ is the decay constant controlling the amount of decay of the weight vector $\mathbf{w}_i(t)$ in each update step.

Similar to the notation $\mathbf{w}$ (4), we can define a vector function $\mathbf{g}$ which augments all the vector functions $\mathbf{g}_1, \mathbf{g}_2, \cdots, \mathbf{g}_m$, i.e.

$$\mathbf{g} = (\mathbf{g}_1^T, \mathbf{g}_2^T, \cdots, \mathbf{g}_m^T)^T. \tag{9}$$

From (6),

$$\begin{aligned} & \mathbf{w}(t+1) - \mathbf{w}(t) \\ = \; & \mu(t) \left\{ (y_t - f(\mathbf{x}_t, \mathbf{w}(t))) \mathbf{g}(\mathbf{x}_t, \mathbf{w}(t)) \right. \\ & \left. - \; \alpha \mathbf{w}(t) \right\}, \end{aligned} \tag{10}$$

Conventionally, the initial weight vector $\mathbf{w}(0)$ is set to a small random vector with elements around zero. The objective function being minimized by this weight decay training algorithm (6) is $\frac{1}{N} \sum_{k=1}^N (y_k - f(\mathbf{x}_k, \mathbf{w}))^2 + \alpha \|\mathbf{w}\|_2^2$, and its convergence is with probability one [49], [50].

III. Injecting node fault during training

Node fault injection-based algorithm is basically a heuristic modification of the weight decay training algorithm by randomly setting zero values to the outputs of the hidden nodes. To model this random node fault injection, we introduce a binary random vector $\mathbf{b}(t) \in \{0, 1\}^m$ at the t^{th} update step.

$$P(b_i(t)) = \begin{cases} 1 - p & \text{if } b_i(t) = 1 \\ p & \text{if } b_i(t) = 0, \end{cases} \tag{11}$$

where the value $0 < p < 1$ is the constant. For all i, j $(i \neq j)$, t, t' $(t \neq t')$, the random variables $b_i(t), b_i(t'), b_j(t)$ are all identical and independent.

To save the space, we denote $\tilde{z}_i(\mathbf{x}_t, \mathbf{a}_i(t), c_i(t))$ by $\tilde{z}_i(t)$. With $b_i(t)$, the output of the i^{th} hidden node, denoted as $\tilde{z}_i$, will be given by

$$\tilde{z}_i(t) = b_i(t) z_i(t). \tag{12}$$

The output of a linear output MLP with node fault injection, denoted as $\tilde{f}$, will be given by

$$\tilde{f}(\mathbf{x}_t, \mathbf{w}(t)) = \sum_{i=1}^{m} d_i(t) b_i(t) z_i(t). \tag{13}$$

On the other hand, we let $\tilde{\mathbf{g}}_i(\mathbf{x}_t, \mathbf{w}_i(t))$ be the perturbed counterpart of $\mathbf{g}_i(\mathbf{x}_t, \mathbf{w}(t))$. By replacing z_i by $\tilde{z}_i$ in (8),

$$\tilde{\mathbf{g}}_i(\mathbf{x}_t, \mathbf{w}(t)) = \begin{bmatrix} \tilde{z}_i(t) \\ d_i(t)\tilde{z}_i(t)(1-\tilde{z}_i(t))\mathbf{x}_t \\ d_i(t)\tilde{z}_i(t)(1-\tilde{z}_i(t)) \end{bmatrix}. \tag{14}$$

Based on (6), (13) and (14), the update equation for $\mathbf{w}_i$ for all $i = 1, \cdots, m$ is defined as follows :

$$\begin{aligned} & \mathbf{w}_i(t+1) - \mathbf{w}_i(t) \\ = \; & \mu(t) \Big\{ (y_t - \tilde{f}(\mathbf{x}_t, \mathbf{w}(t))) \tilde{\mathbf{g}}_i(\mathbf{x}_t, \mathbf{w}(t)) \\ & - \alpha \mathbf{w}_i(t) \Big\}, \end{aligned} \tag{15}$$

where $\mu(t) > 0$ is the step size at the t^{th} step and $\alpha > 0$ is called the decay constant. Similar to (10), (15) can be rewritten as follows :

$$\begin{aligned} & \mathbf{w}(t+1) - \mathbf{w}(t) \\ = \; & \mu(t) \Big\{ (y_t - \tilde{f}(\mathbf{x}_t, \mathbf{w}(t))) \tilde{\mathbf{g}}(\mathbf{x}_t, \mathbf{w}(t)) \\ & - \alpha \mathbf{w}(t) \Big\}, \end{aligned} \tag{16}$$

where $\tilde{\mathbf{g}}$ is a vector function augmenting $\tilde{\mathbf{g}}_1, \tilde{\mathbf{g}}_2, \cdots, \tilde{\mathbf{g}}_m$, i.e.

$$\tilde{\mathbf{g}} = (\tilde{\mathbf{g}}_1^T, \tilde{\mathbf{g}}_2^T, \cdots, \tilde{\mathbf{g}}_m^T)^T. \tag{17}$$

By using notation $\delta_k(t)$, an alternative form for (15) is given by

$$\begin{aligned} & \mathbf{w}_i(t+1) - \mathbf{w}_i(t) \\ = \; & \mu(t) \sum_{k=1}^{M} \delta_k(t) (y_k - \tilde{f}(\mathbf{x}_k, \mathbf{w}(t))) \tilde{\mathbf{g}}_i(\mathbf{x}_k, \mathbf{w}(t)) \\ & - \alpha \mathbf{w}_i(t). \end{aligned} \tag{18}$$

This form is particular useful for the derivations of the mean update equation and the objective function of the on-line training algorithms.

IV. Objective function

Since $b_i(t)$ and $b_j(t)$ are independent for all $i \neq j$, the expectation of $b_i(t)b_j(t)$ denoted by $E[b_i(t)b_j(t)]$ is given by

$$E[b_i(t)b_j(t)] = \begin{cases} (1-p) & \text{if } i = j, \\ (1-p)^2 & \text{if } i \neq j. \end{cases} \tag{19}$$

For given $\mathbf{x}_t$ and y_t, one can show by using (19) that

$$E[y_t \tilde{z}_i(t)] = (1-p) y_t z_i(t), \tag{20}$$

$$E[y_t \tilde{z}_i(t)(1-\tilde{z}_i(t))] = (1-p) y_t z_i(t)(1-z_i(t)). \tag{21}$$

As

$$\begin{aligned} & \tilde{f}(\mathbf{x}_t, \mathbf{w}(t)) \tilde{z}_i(t) \\ = \; & \left(\sum_{j=1}^{m} d_j(t) b_j(t) z_j(t) \right) b_i(t) z_i(t), \\ = \; & b_i(t)^2 d_i(t) z_i^2(t) \\ & + \sum_{j=1, j\neq i}^{m} b_i(t) b_j(t) d_j(t) z_j(t) z_i(t). \end{aligned}$$

Hence,

$$\begin{aligned} & E[\tilde{f}(\mathbf{x}_t, \mathbf{w}(t)) \tilde{z}_i(t)] \\ = \; & (1-p) d_i(t) z_i^2(t) \\ & + (1-p)^2 \sum_{j=1, j\neq i}^{m} d_j(t) z_j(t) z_i(t). \\ = \; & (1-p) z_i(t) \left\{ p d_i(t) z_i(t) + (1-p) \sum_{j=1}^{m} d_j(t) z_j(t) \right\} \\ = \; & (1-p) z_i(t) \left\{ p\, d_i(t) z_i(t) + (1-p) f(\mathbf{x}_t, \mathbf{w}(t)) \right\} \end{aligned} \tag{22}$$

and similarly

$$\begin{aligned} & E[\tilde{f}(\mathbf{x}_t, \mathbf{w}(t)) \tilde{z}_i(t)(1-\tilde{z}_i(t))] \\ = \; & (1-p) z_i(t)(1-z_i(t)) \\ & \times \left\{ p\, d_i(t) z_i(t) + (1-p) f(\mathbf{x}_t, \mathbf{w}(t)) \right\}. \end{aligned} \tag{23}$$

Applying (20) - (23) and assuming that each sample in the dataset $\mathcal{D}$ has equal probability to be selected in each step, the expectation of (15) over all random variables $b_1(t), \cdots, b_m(t)$ and $\delta_1(t) \cdots, \delta_N(t)$ for a given $\mathbf{w}(t)$ can be expressed as follows :

$$\begin{aligned} & E[\mathbf{w}_i(t+1) | \mathbf{w}(t)] \\ = \; & \mathbf{w}_i(t) + \frac{\mu'(t)}{N} \sum_{k=1}^{N} (y_k - f(\mathbf{x}_k, \mathbf{w}(t))) \mathbf{g}_i(\mathbf{x}_k, \mathbf{w}(t)) \\ - \; & \frac{\mu'(t) p}{N} \sum_{k=1}^{N} \left[\sum_{j\neq i} d_j(t) z_j(t) \right] \mathbf{g}_i(\mathbf{x}_k, \mathbf{w}(t)) \\ - \; & \mu'(t) \alpha' \mathbf{w}_i(t), \end{aligned} \tag{24}$$

where $\mu'(t) = (1-p)\mu(t)$ and $\alpha' = \alpha/(1-p)$.

The mean update equation for (24) can be written as follows :

$$E[\mathbf{w}_i(t+1)|\mathbf{w}(t)] = \mathbf{w}_i(t) - \mu'(t)\ \frac{\partial}{\partial \mathbf{w}_i} V(\mathbf{w})\Big|_{\mathbf{w}=\mathbf{w}(t)}. \quad (25)$$

In which, the scalar function $V(\mathbf{w})$ is called the objective function of the algorithm (24). Let $\frac{\partial}{\partial \mathbf{w}} V(\mathbf{w})$ denotes the vector augmenting $\frac{\partial}{\partial \mathbf{w}_1} V(\mathbf{w})$ $\frac{\partial}{\partial \mathbf{w}_2} V(\mathbf{w})$, $\cdots$, $\frac{\partial}{\partial \mathbf{w}_m} V(\mathbf{w})$, i.e.

$$\frac{\partial}{\partial \mathbf{w}} V(\mathbf{w}) = \left(\frac{\partial}{\partial \mathbf{w}_1} V(\mathbf{w})^T, \cdots, \frac{\partial}{\partial \mathbf{w}_m} V(\mathbf{w})^T \right)^T. \quad (26)$$

Equation (25) can be written as follows :

$$E[\mathbf{w}(t+1)|\mathbf{w}(t)] = \mathbf{w}(t) - \mu'(t)\ \frac{\partial}{\partial \mathbf{w}} V(\mathbf{w})\Big|_{\mathbf{w}=\mathbf{w}(t)}. \quad (27)$$

Theorem 1: The objective function of algorithm (24) is given by

$$\begin{aligned} V(\mathbf{w}) &= \frac{1}{2N} \sum_{k=1}^{N} (y_k - f(\mathbf{x}_k, \mathbf{w}))^2 \\ &+ \frac{p}{2N} \sum_{k=1}^{N} \mathbf{d}^T \left(\mathbf{G}(\mathbf{x}_k, \mathbf{w}) - \mathbf{H}(\mathbf{x}_k, \mathbf{w}) \right) \mathbf{d} \\ &+ \frac{\alpha}{2(1-p)} \|\mathbf{w}\|_2^2. \end{aligned} \quad (28)$$

The matrices $\mathbf{G}(\mathbf{x}_k, \mathbf{w})$ *and* $\mathbf{H}(\mathbf{x}_k, \mathbf{w})$ *in (28) are given by*

$$\mathbf{G}(\mathbf{x}_k, \mathbf{w}) = \mathbf{diag}\{z_1^2(\mathbf{x}_k, \mathbf{w}), \cdots, z_m^2(\mathbf{x}_k, \mathbf{w})\}, \quad (29)$$

$$\mathbf{H}(\mathbf{x}_k, \mathbf{w}) = \mathbf{z}(\mathbf{x}_k, \mathbf{w}) \mathbf{z}(\mathbf{x}_k, \mathbf{w})^T, \quad (30)$$

and $\|\mathbf{w}\|_2^2 = \mathbf{w}^T \mathbf{w}$.

(Proof) Taking derivative of (28) with respect to $\mathbf{w}_i$ and then setting $\mathbf{w}_i$ to $\mathbf{w}_i(t)$, one can show that (24) is equivalent to (25). **Q.E.D.**

By (15), (24), (25) and (28), it is clear that (15) is a stochastic gradient descent algorithm which minimizes the objective function $V(\mathbf{w})$. One should also note that the factor

$$\frac{p}{N} \sum_{k=1}^{N} \mathbf{d}^T \left(\mathbf{G}(\mathbf{x}_k, \mathbf{w}) - \mathbf{H}(\mathbf{x}_k, \mathbf{w}) \right) \mathbf{d} \quad (31)$$

in (28) is similar to the fault tolerant regularizer derived in [28]. This regularizer has distinctive property in comparison with the regularizers proposed for improving generalization [9], [47]. Since the diagonal elements of the matrix $\mathbf{G} - \mathbf{H}$ are all zeros and the matrix is symmetric, the summation of all its eigenvalues must be zero and the eigenvalues must be real. Thus, it is not possible that all the eigenvalues are of the same sign, or that all the eigenvalues are zeros. Some eigenvalues are positive and some eigenvalues are negative. That means, for fault tolerant, the matrix must contain some positive eigenvalues and some negative eigenvalues. In other words, the effect of the additional term (31) is not simply penalizing the weight magnitudes. It could magnify weight magnitude as well.

V. Convergence of Algorithm

To accomplish the proof of convergence, we need the following definitions for absolute value and norms. Let γ be any real number in R; and $\mathbf{q} = (q_1, q_2, \cdots, q_v)^T$ be a vector in R^v. The absolute value of a number γ is denoted as $|\gamma|$. The l_∞ norm and l_2 norm of vector $\mathbf{q}$ are denoted as $\|\mathbf{q}\|_\infty$ and $\|\mathbf{q}\|_2$, where $\|\mathbf{q}\|_\infty = \max\{|q_1|, \cdots, |q_v|\}$, and $\|\mathbf{q}\|_2 = \left[\sum_{i=1}^{m} q_i^2\right]^{1/2}$. By the definitions of l_∞ and l_2 norms,

$$\|\mathbf{q}\|_2 \leq \sqrt{m}\, \|\mathbf{q}\|_\infty. \quad (32)$$

The convergence proof will be presented in the rest of this section. We first show that $\|\mathbf{w}(t)\|_2$ is bounded for all t. Then, we apply a theorem in [49] to show that the convergence of the algorithm (15).

A. Boundedness of $\|\mathbf{w}(t)\|_2$

The boundedness of $\|\mathbf{w}(t)\|_2$ will be explained one by one from boundedness of $\|\mathbf{d}(t)\|_2$, $\|\mathbf{a}_i(t)\|_2$ and $|c_i(t)|$ for all $i = 1, \cdots, m$.

By (3), (14) and (15), the update of $d_i(t)$ can be expressed as follows :

$$\begin{aligned} & d_i(t+1) - d_i(t) \\ = \ & \mu(t) \left\{ (y_t - \mathbf{d}^T(t) \tilde{\mathbf{z}}(t)) \tilde{z}_i(t) - \alpha\, d_i(t) \right\}. \end{aligned} \quad (33)$$

In vector-matrix form,

$$\begin{aligned} & \mathbf{d}(t+1) - \mathbf{d}(t) \\ = \ & \mu(t) \left\{ (y_t - \mathbf{d}^T(t) \tilde{\mathbf{z}}(t)) \tilde{\mathbf{z}}(t) - \alpha\, \mathbf{d}(t) \right\}. \end{aligned} \quad (34)$$

Let

$$\mathbf{B}(t) = (1 - \mu(t)\alpha) I_{m \times m} - \mu(t) \tilde{\mathbf{z}}(t) \tilde{\mathbf{z}}^T(t).$$

$$\mathbf{d}(t+1) = \mathbf{B}(t) \mathbf{d}(t) + \mu(t) y_t \tilde{\mathbf{z}}(t). \quad (35)$$

It can be shown that the eigenvalues of $\tilde{\mathbf{z}}(t)\tilde{\mathbf{z}}^T(t)$ are 0 and $\sum_{i=1}^{m} \tilde{z}_i(t)^2$ (see Lemma 6.3 in [50]). Thus, the eigenvalues of the matrix $\mathbf{B}(t)$ in (35) are

$$1 - \mu(t)\alpha \quad \text{and} \quad 1 - \mu(t)\alpha - \mu(t) \sum_{i=1}^{m} \tilde{z}_i(t)^2.$$

As $0 < \tilde{z}_i(t) < 1$, one can set the step size small enough to ensure that the eigenvalues of $\mathbf{B}(t)$ is positive and less than one. For instance, $0 < \mu(t) < \frac{1}{\alpha + m}$. In such case $(1 - \mu(t)\alpha)$ will be the largest eigenvalue of $\mathbf{B}(t)$.

By the definition of l_∞ norm, and the facts that $|y_t| < \infty$ and $0 < \tilde{z}_i(t) < 1$,

$$\|\mathbf{d}(t+1)\|_\infty \leq (1 - \mu(t)\alpha) \|\mathbf{d}(t)\|_\infty + \mu(t)\kappa_1, \quad (36)$$

where $\kappa_1 = \max |y_t|$. The following lemma states the boundedness of $\|\mathbf{d}(t)\|_2$.

Lemma 1: For all $t \geq 0$, the l_2 norm of $\mathbf{d}(t)$ is bounded, and is given by

$$\|\mathbf{d}(t)\|_2 \leq \kappa_2, \quad (37)$$

where $\kappa_2 = \max\{\sqrt{m}\, \kappa_1 / \alpha, \sqrt{m} \|\mathbf{d}(0)\|_\infty\}$.

(Proof) We consider the following cases: 1) the initial condition of $\|\mathbf{d}(0)\|_\infty \geq \kappa_1/\alpha$; and 2) the initial condition $\|\mathbf{d}(0)\|_\infty < \kappa_1/\alpha$. For each case, we prove by induction that the condition (37) holds for all $t \geq 0$.

Case 1: $\|\mathbf{d}(0)\|_\infty \geq \kappa_1/\alpha$ implies

$$\begin{aligned} \kappa_2 &= \sqrt{m}\|\mathbf{d}(0)\|_\infty, && (38)\\ \kappa_1 &\leq \alpha\|\mathbf{d}(0)\|_\infty. && (39)\end{aligned}$$

Hence by (39), (36) can be rewritten as follows :

$$\|\mathbf{d}(t+1)\|_\infty \leq (1-\mu(t)\alpha)\|\mathbf{d}(t)\|_\infty + \mu(t)\alpha\|\mathbf{d}(0)\|_\infty. \quad (40)$$

Now, we are able to show by induction that $\|\mathbf{d}(t)\|_2 \leq \kappa_2$.

For $t = 0$ in (40),

$$\begin{aligned} \|\mathbf{d}(1)\|_\infty &\leq (1-\mu(0)\alpha)\|\mathbf{d}(0)\|_\infty + \mu(0)\alpha\|\mathbf{d}(0)\|_\infty,\\ &= \|\mathbf{d}(0)\|_\infty.\end{aligned}$$

Thus by (32),

$$\begin{aligned} \|\mathbf{d}(1)\|_2 &\leq \sqrt{m}\|\mathbf{d}(1)\|_\infty,\\ &\leq \sqrt{m}\|\mathbf{d}(0)\|_\infty,\\ &= \kappa_2.\end{aligned}$$

Condition (37) holds for $t = 0$.

For $t = 1$ in (40),

$$\begin{aligned} \|\mathbf{d}(2)\|_\infty &\leq (1-\mu(1)\alpha)\|\mathbf{d}(1)\|_\infty + \mu(1)\alpha\|\mathbf{d}(0)\|_\infty,\\ &\leq (1-\mu(1)\alpha)\|\mathbf{d}(0)\|_\infty + \mu(1)\alpha\|\mathbf{d}(0)\|_\infty,\\ &= \|\mathbf{d}(0)\|_\infty.\end{aligned}$$

Thus by (32) and (38),

$$\begin{aligned} \|\mathbf{d}(2)\|_2 &\leq \sqrt{m}\|\mathbf{d}(2)\|_\infty,\\ &\leq \sqrt{m}\|\mathbf{d}(0)\|_\infty,\\ &= \kappa_2.\end{aligned}$$

Condition (37) holds for $t = 1$ as well.

Next, let the condition (37) holds for $t = t'$. For $t = t'+1$,

$$\begin{aligned} \|\mathbf{d}(t'+1)\|_\infty &\leq (1-\mu(t')\alpha)\|\mathbf{d}(t')\|_\infty + \mu(t')\alpha\|\mathbf{d}(0)\|_\infty,\\ &\leq (1-\mu(t')\alpha)\|\mathbf{d}(0)\|_\infty + \mu(t')\alpha\|\mathbf{d}(0)\|_\infty,\\ &= \|\mathbf{d}(0)\|_\infty.\end{aligned}$$

Thus by (32) and (38),

$$\begin{aligned} \|\mathbf{d}(t'+1)\|_2 &\leq \sqrt{m}\|\mathbf{d}(t'+1)\|_\infty,\\ &\leq \sqrt{m}\|\mathbf{d}(0)\|_\infty,\\ &= \kappa_2.\end{aligned}$$

Condition (37) holds for $t = t'+1$.

By the principle of mathematical induction, if $\|\mathbf{d}(0)\|_\infty \geq \kappa_1/\alpha$ condition (37) holds for all $t \geq 0$.

Case 2: $\|\mathbf{d}(0)\|_\infty < \kappa_1/\alpha$ implies that

$$\kappa_2 = \sqrt{m}\,\kappa_1/\alpha. \quad (41)$$

For $t = 0$ in (36),

$$\begin{aligned} \|\mathbf{d}(1)\|_\infty &< (1-\mu(0)\alpha)\|\mathbf{d}(0)\|_\infty + \mu(0)\kappa_1,\\ &< (1-\mu(0)\alpha)\frac{\kappa_1}{\alpha} + \mu(0)\kappa_1,\\ &= \frac{\kappa_1}{\alpha}.\end{aligned}$$

Thus by (32) and (41),

$$\begin{aligned} \|\mathbf{d}(1)\|_2 &< \sqrt{m}\|\mathbf{d}(1)\|_\infty,\\ &< \frac{\sqrt{m}\,\kappa_1}{\alpha},\\ &= \kappa_2.\end{aligned}$$

Condition (37) holds for $t = 0$.

For $t = 1$ in (36),

$$\begin{aligned} \|\mathbf{d}(2)\|_\infty &< (1-\mu(1)\alpha)\|\mathbf{d}(1)\|_\infty + \mu(1)\kappa_1,\\ &< (1-\mu(1)\alpha)\frac{\kappa_1}{\alpha} + \mu(1)\kappa_1,\\ &= \frac{\kappa_1}{\alpha}.\end{aligned}$$

Thus by (32) and (41),

$$\begin{aligned} \|\mathbf{d}(2)\|_2 &< \sqrt{m}\|\mathbf{d}(2)\|_\infty,\\ &< \frac{\sqrt{m}\,\kappa_1}{\alpha},\\ &= \kappa_2.\end{aligned}$$

Condition (37) holds for $t = 1$.

Next, let the condition (37) holds for $t = t'$. For $t = t'+1$,

$$\begin{aligned} \|\mathbf{d}(t'+1)\|_\infty &< (1-\mu(t')\alpha)\|\mathbf{d}(t')\|_\infty + \mu(t')\kappa_1,\\ &< (1-\mu(t')\alpha)\frac{\kappa_1}{\alpha} + \mu(t')\kappa_1,\\ &= \frac{\kappa_1}{\alpha}.\end{aligned}$$

Thus by (32) and (41),

$$\begin{aligned} \|\mathbf{d}(t'+1)\|_2 &< \sqrt{m}\|\mathbf{d}(t'+1)\|_\infty,\\ &< \frac{\sqrt{m}\,\kappa_1}{\alpha},\\ &= \kappa_2.\end{aligned}$$

Condition (37) holds for $t = t'+1$.

By the principle of mathematical induction, if $\|\mathbf{d}(0)\|_\infty < \kappa_1/\alpha$ condition (37) holds for all $t \geq 0$.

Since condition (37) holds for both cases when $\|\mathbf{d}(0)\|_\infty \geq \kappa_1/\alpha$ and $\|\mathbf{d}(0)\|_\infty < \kappa_1/\alpha$, we can conclude that $\|\mathbf{d}(t)\|_2 \leq \max\{\sqrt{m}\,\kappa_1/\alpha, \sqrt{m}\|\mathbf{d}(0)\|_\infty\}$ for all $t \geq 0$. The proof is completed. **Q.E.D.**

By (3), (14) and (15), the update of $\mathbf{a}_i(t)$ can be expressed as follows :

$$\begin{aligned} &\mathbf{a}_i(t+1)\\ =\ &\mathbf{a}_i(t) + \mu(t)(y_t - \mathbf{d}^T(t)\tilde{\mathbf{z}}(t))\tilde{z}_i(t)(1-\tilde{z}_i(t))\mathbf{x}_t\\ &- \mu(t)\alpha\,\mathbf{a}_i(t). \quad (42)\\ =\ &(1-\mu(t)\alpha)\mathbf{a}_i(t)\\ &+ \mu(t)(y_t - \mathbf{d}^T(t)\tilde{\mathbf{z}}(t))\tilde{z}_i(t)(1-\tilde{z}_i(t))\mathbf{x}_t. \quad (43)\end{aligned}$$

As $|y_t|$, $\|\mathbf{x}_t\|_\infty$ and $\|\tilde{\mathbf{z}}(t)\|_\infty$ are bounded; and the proof in Lemma 1 has showed that $\|\mathbf{d}(t)\|_\infty$ are bounded,

$$\kappa_3 = \max\left\{\|(y_t - \mathbf{d}^T(t)\tilde{\mathbf{z}}(t))\tilde{z}_i(t)(1-\tilde{z}_i(t))\mathbf{x}_t\|_\infty\right\}.$$

Thus,

$$\|\mathbf{a}_i(t+1)\|_\infty \le (1-\mu(t)\alpha)\|\mathbf{a}_i(t)\|_\infty + \mu(t)\kappa_3. \quad (44)$$

Applying the same technique as the boundedness proof of $\mathbf{d}(t)$, we state without proof the following lemma on the boundedness of $\|\mathbf{a}_i(t)\|_2$.

Lemma 2: For all $i = 1, \cdots, m$, the l_2 norm of $\mathbf{a}_i(t)$ is bounded for all $t \ge 0$ and is given by

$$\|\mathbf{a}_i(t)\|_2 \le \kappa_5, \quad (45)$$

where $\kappa_5 = \max\{\sqrt{n}\kappa_3/\alpha, \sqrt{n}\|\mathbf{a}_i(0)\|_\infty\}$.

Also by (3), (14) and (15), the update of $c_i(t)$ can be expressed as follows :

$$\begin{aligned} & c_i(t+1) \\ = \; & (1-\mu(t)\alpha)c_i(t) \\ & + \mu(t)(y_t - \mathbf{d}^T(t)\tilde{\mathbf{z}}(t))\tilde{z}_i(t)(1-\tilde{z}_i(t)). \end{aligned} \quad (46)$$

As $|y_t|$, $\|\tilde{\mathbf{z}}(t)\|_\infty$ and $\|\mathbf{d}(t)\|_\infty$ are bounded,

$$\kappa_4 = \max\left\{\left|(y_t - \mathbf{d}^T(t)\tilde{\mathbf{z}}(t))\tilde{z}_i(t)(1-\tilde{z}_i(t))\right|\right\}.$$

Thus,

$$|c_i(t+1)| \le (1-\mu(t)\alpha)|c_i(t)| + \mu(t)\kappa_4. \quad (47)$$

Applying the same technique as the boundedness proof of $\mathbf{d}(t)$, we state without proof the following lemma on the boundedness of $|c_i(t)|$.

Lemma 3: For all $i = 1, \cdots, m$, $|c_i(t)|$ is bounded for all $t \ge 0$ and is given by

$$|c_i(t)| \le \kappa_6, \quad (48)$$

where $\kappa_6 = \max\{\kappa_4/\alpha, |c_i(0)|\}$.

By Lemma 1-3, we can show the boundedness of $\|\mathbf{w}(t)\|_2$, as stated in the following theorem.

Theorem 2: $\|\mathbf{w}(t)\|_2 \le \kappa_7$ for all $t \ge 0$, where

$$\kappa_7 = \sqrt{(n+2)m}\,\max\{\kappa_1/\alpha, \kappa_3/\alpha, \kappa_4/\alpha, \|\mathbf{w}(0)\|_\infty\}.$$

(Proof) Since $\|\mathbf{w}(t)\|_2 \le \sqrt{(n+2)m}\|\mathbf{w}(t)\|_\infty$ and

$$\begin{aligned} \|\mathbf{w}(t)\|_\infty \le \; & \max\{\kappa_1/\alpha, \|\mathbf{d}(0)\|_\infty, \\ & \kappa_3/\alpha, \|\mathbf{a}_1(0)\|_\infty, \\ & \cdots, \|\mathbf{a}_m(0)\|_\infty, \kappa_4/\alpha, \\ & |c_1(0)|, \cdots, |c_m(0)|\}, \end{aligned}$$

$$\begin{aligned} \|\mathbf{w}(t)\|_2 &\le \sqrt{(n+2)m}\,\max\{\kappa_1/\alpha, \kappa_3/\alpha, \\ &\quad \kappa_4/\alpha, \|\mathbf{w}(0)\|_\infty\} \\ &= \kappa_7. \end{aligned}$$

The proof is completed. **Q.E.D.**

B. Convergence proof

Now, we can proceed to the proof of convergence. Before that, we need the following assumption on a nonnegative number sequence $\gamma(t)$.

Assumption 1: $\gamma(t)$ is nonnegative for all t and it is a decreasing sequence fulfilling the following conditions.

$$\sum_t \gamma(t) = \infty, \quad \sum_t \gamma^2(t) < \infty. \quad (49)$$

The convergence proof follows the same approach as in [50], which applies the following theorem from H. White in [49].

Lemma 4 ([49]): Let $\{\mathbf{Z}(t)\}_{t\ge 0}$ be a sequence of bounded i.i.d. random vectors, and $\mathbf{M} : R^l \times R^v \to R^v$ a continuously differentiable function. Suppose that, (i) for each $\mathbf{w} \in R^l$, the expectation

$$\bar{\mathbf{M}}(\mathbf{w}) \equiv E[\mathbf{M}(\mathbf{Z}(t), \mathbf{w})] \quad (50)$$

is bounded, (ii) there exists a twice continuously differentiable function $\mathcal{L} : R^l \to R$ such that

$$\left(\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) \le 0 \quad (51)$$

for all $\mathbf{w} \in R^l$, and (iii) $\{\gamma(t)\}_{t=1}^\infty$ is a sequence satisfying assumptions (49). Define the sequence $\{\mathbf{w}(t)\}_{t\ge 0}$ iteratively by

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \gamma(t)\mathbf{M}(\mathbf{Z}(t), \mathbf{w}(t)), \quad (52)$$

where $\mathbf{w}(0)$ is arbitrarily given. Then either $\mathbf{w}(t)$ converges to $\{\mathbf{w}|\left(\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) = 0\}$ or $\mathbf{w}(t)$ diverges with probability 1.

Now, we can state the convergence of the algorithm (15) in the following theorem.

Theorem 3: For arbitrarily given $\mathbf{w}(0)$, $\mathbf{w}(t)$ defined by (15) converges to $\{\mathbf{w}|\left(\frac{\partial}{\partial \mathbf{w}}V(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) = 0\}$ with probability 1.

(Proof) Compare (16) and (52), we can define

$$\mathbf{Z}(t) = (\mathbf{x}_t^T, y_t, \mathbf{b}(t)^T)^T, \quad (53)$$

$$\mathbf{M}(\mathbf{Z}(t), \mathbf{w}) = (y_t - \tilde{f}(\mathbf{x}_t, \mathbf{w}))\tilde{\mathbf{g}}(\mathbf{x}_t, \mathbf{w}) - \alpha\mathbf{w}, \quad (54)$$

$$\gamma(t) = \mu'(t). \quad (55)$$

Condition (i): As the elements in the vector $\mathbf{Z}(t) = (\mathbf{x}_t^T, y_t, \mathbf{b}(t)^T)^T$ are bounded and Lemma 1 to 3 show that $\mathbf{w}(t)$ is bounded for all t, it implies that $E[\mathbf{M}(\mathbf{Z}(t), \mathbf{w})]$ is bounded. Condition (i) holds.

Condition (ii): From (27), it is clear that

$$\begin{aligned} \bar{M}(\mathbf{w}) &= E[\mathbf{M}(\mathbf{Z}(t), \mathbf{w})] \\ &= -\frac{\partial}{\partial \mathbf{w}}V(\mathbf{w}). \end{aligned}$$

So, we can define that

$$\mathcal{L}(\mathbf{w}) = V(\mathbf{w}). \quad (56)$$

From (28), one can show that $V(\mathbf{w})$ twice differentiable. Thus, $\mathcal{L}(\mathbf{w})$ is twice differentiable. Besides,

$$\left(\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) = -\frac{\partial V(\mathbf{w})}{\partial \mathbf{w}}^T \frac{\partial V(\mathbf{w})}{\partial \mathbf{w}} \leq 0.$$

So, Condition (ii) holds.

Condition (iii): For the step size $\mu'(t)$, one can define it in a way to satisfy the conditions stated in (49). For instance, one can set $\mu'(t) \propto t^{-1}$. Condition (iii) holds.

As Condition (i), (ii) and (iii) hold, it is concluded by Lemma 4 that $\mathbf{w}(t)$ defined by (15) converges to $\{\mathbf{w}|\left(\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) = 0\}$ with probability 1 or $\mathbf{w}(t)$ diverges with probability one.

However, Lemma 1 to 3 have showed that $\mathbf{w}(t)$ is bounded for all t. In other words, $\mathbf{w}(t)$ cannot diverge.

Therefore, $\mathbf{w}(t)$ defined by (15) must converge to $\{\mathbf{w}|\left(\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) = 0\}$ (or equivalently $\{\mathbf{w}|\left(\frac{\partial}{\partial \mathbf{w}}V(\mathbf{w})\right)^T \bar{\mathbf{M}}(\mathbf{w}) = 0\}$) with probability 1. The proof is completed. **Q.E.D.**

VI. Conclusion

In this paper, we have presented an on-line node fault injection training algorithm (15) for MLPs. Thus, we have shown that the weight vectors ($\mathbf{w}(t)$ for all $t \geq 0$) obtained by this algorithm is always bounded. With this boundedness condition, we have applied a theorem from H.White (Lemma 4) and shown that the convergence of algorithm (15) is with probability one (Theorem 3). Besides, the objective function being minimized by this algorithm has been found, as stated in (28). The equivalence between this objective function and the one defined in [28] is remarked.

Acknowledgement

The research work done in this paper is supported in part by National Science Council, Taiwan, under Research Grants 97-2221-E-005-050 and 98-2221-E-005-048.

References

[1] An G. The effects of adding noise during backpropagation training on a generalization performance, *Neural Computation*, Vol.8, 643-674, 1996.

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

[3] 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.

[4] 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.

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

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

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

[8] 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.

[9] Chen S., Local regularization assisted orthogonal least squares regression, *Neurocomputing*, pp. 559-585, 2006.

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

[11] Deodhare D., M. Vidyasagar and S. Sathiya Keerthi, Synthesis of fault-tolerant feedforward neural networks using minimax optimization, *IEEE Transactions on Neural Networks*, Vol.9(5), 891-900, 1998.

[12] Edwards P.J. and A.F. Murray, Fault tolerant via weight noise in analog VLSI implementations of MLP's – A case study with EPSILON, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Vol.45, No.9, p.1255-1262, Sep 1998.

[13] Grandvalet Y., S. Canu, A comment on noise injection into inputs in back-propagation learning, *IEEE Transactions on Systems, Man, and Cybernetics*, 1995.

[14] Grandvalet Y., S. Canu and S. Boucheron, Noise injection: Theoretical prospects, *Neural Computation*, Vol. 9, 1093-1107, 1997.

[15] Hammadi N.C. and I. Hideo, A learning algorithm for fault tolerant feedforward neural networks, *IEICE Transactions on Information & Systems*, Vol. E80-D, No.1, 1997.

[16] 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.

[17] S. Himavathi, D. Anitha and A. Muthuramalingam, Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization, *IEEE Transactions on Neural Networks*, Vol.18, 880-888, 2007.

[18] Ho K., C.S. Leung, and J. Sum, On weight-noise-injection training, M.Koeppen, N.Kasabov and G.Coghill (Eds.), *Advances in Neuro-Information Processing*, Springer LNCS 5507, pp. 919-926, 2009.

[19] K. Ho, C.S. Leung and J. Sum, Convergence and Objective Functions of Some Fault/Noise Injection-Based online Learning Algorithms for RBF Networks, *IEEE Transactions on Neural Networks*, in press.

[20] Jiang Y. *et al*, A study of the effect of noise injection on the training of artificial neural networks, *Proc. IJCNN'09*, 2009.

[21] 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.

[22] Judd S. and P. W. Munro, Nets with unreliable hidden nodes learn error-correcting codes, in *Advances in Neural Information Proceesing Systems*, Vol.5, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. San Mateo, CA: Morgan Kanfmanu, pp. 89-96, 1993.

[23] Kamiura N., *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.

[24] LeCun Y. *et al.*, Optimal brain damage, *Advances in Neural Information Processing Systems 2* (D.S. Touretsky, ed.) 396-404, 1990.

[25] Leung C.S., K.W. Wong, P.F. Sum and L.W. Chan, A pruning method for recursive least squared algorithm, *Neural Networks*, 14:147-174, 2001.

[26] 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.

[27] Leung C.S., K.W.Wong, J. Sum, and L.W.Chan, On-line training and pruning for RLS algorithms, *Electronics Letters*, Vol.32, No.23, 2152-2153, 1996.

[28] Leung C.S., J. Sum, A fault tolerant regularizer for RBF networks, *IEEE Transactions on Neural Networks*, Vol. 19 (3), pp.493-507, 2008.

[29] Leung C.S. and J. Sum Analysis on generalization error of faulty RBF networks with weight decay regularizer, in *Proceedings of ICONIP 2008*, Springer LNCS, 2009.

[30] Matsuoka K., Noise injection into inputs in back-propagation learning, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.22(3), 436-440, 1992.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] Antony W. Savich, Medhat Moussa and Shawki Areibi, The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study, *IEEE Transactions on Neural Networks*, Vol.18, 240-252, 2007.

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

[41] Sugiyama, M. and Ogawa, H., Optimal design of regularization term and regularization parameter by subspace information criterion, *Neural Networks*, Vol.15, 349-361, 2002.

[42] Sum J., C.S. Leung, L. Hsu, Y. Huang, An objective function for single node fault RBF learning, in *Proceedings of TAAI'2007*.

[43] Sum J., C.S. Leung and K. Ho, On objective function, regularizer and prediction error of a learning algorithm for dealing with multiplicative weight noise, *IEEE Transactions on Neural Networks* Vol.20(1), Jan, 2009.

[44] Sum J., C.S. Leung, and K. Ho, On node-fault-injection training an RBF network, M.Koeppen, N.Kasabov and G.Coghill (Eds.), *Advances in Neuro-Information Processing*, Springer LNCS 5507, pp. 324-331, 2009.

[45] Sum J. and K. Ho, SNIWD: Simultaneous weight noise injection with weight decay for MLP training, in *Proceedings of ICONIP 2009*, Springer LNCS 5863, 2009.

[46] Takanami I., M. Sato and Y. P. Yang, A fault-value injection approach for multiple-weight-fault tolerance of MNNs, *Proc. of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, vol. III, pp. 515-520, 2000.

[47] Tipping M.E., The relevance vector machine, in *Advances in Neural Information Processing Systems*, Vol.12. p.652-658, Cambridge, MA: MIT Press, 2000.

[48] Velazco R. *et al*, SEU fault tolerance in artificial neural networks, *IEEE Transactions on Nuclear Science*, Vol. 42 (6), 1856-1862, 1995.

[49] White H., Some asymptotic results for learning in single hidden-layer feedforward network models, *Journal of American Statistical Associtation*, Vol.84, No.408, p.1003-1013, December, 1989.

[50] Zhang, Huisheng, Wei Wu, Fei Liu, and Mingchen Yao, Boundedness and convergence of online gradient method with penalty for feedforward neural networks *IEEE Transactions on Neural Networks*, Vol.20(6), 1050-1054, June 2009.

[51] Zhu J.H. and P. 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.