Proc. Natl. Sci. Counc. ROC(A)
Vol. 23, No. 5, 1999. pp. 599-608

(Scientific Note)

Fault Tolerant Training of Feedforward Neural Networks

BUH-YUN SHER* AND WEN-SHONG HSIEH**

*Institute of Electrical Engineering
National Sun Yat-Sen University
Kaoshung, Taiwan, R.O.C.
**Institute of Information and Computer Engineering
National Sun Yat-Sen University
Kaoshung, Taiwan, R.O.C.

(Received May 27, 1998; Accepted December 19, 1998)

ABSTRACT

Fault tolerance is one of the key performance measures of artificial neural networks (ANN's) and is often viewed as an inherent feature of ANN's. But without precise designing, it is not able to guarantee the degree of fault tolerance. This paper presents an extensive study on the fault tolerant property of feedforward neural networks. We propose a constraint backpropagation (CBP) training method, which can guarantee a high degree of fault tolerance when one or two hidden nodes fail. In order to achieve the goal of fault tolerance, we define an energy term, called constraint energy, that measures the performance degradation when some hidden nodes fail. During training, both the normal energy and the constraint energy will be minimized. We also develop a simple technique called output node saturation (ONS). By incorporating CBP with ONS, we can find a network which maintains exactly the same performance as a normal network when some hidden nodes fail. Experimental results show that a network trained by CBP also possess better generalization properties than does one trained by normal backpropagation (BP).

Key Words: feedforward networks, fault tolerance, backpropagation, output nodes saturation, constraint backpropagation

I. Introduction

Fully connected feedforward neural networks trained by backpropagation (BP) or by many of its variants have been most popular for pattern recognition, data mining, and many other application fields (Arun and Polycarpou, 1997; Abhijit and Macy, 1996; Diamantaras and Kung, 1996). BP training can be viewed as an optimization process of a criterion function with respect to linked weight values. The mean squared error is the only criterion that backpropagation training to be optimized, thus some other performance measures, such as fault tolerance and the generalization capability, are taken only as side effects. Neither the property of fault tolerance nor generalization can be controlled during the training phase.

Fault tolerance is one of the frequently cited features of artificial neural networks. It has been claimed that neural networks are inherently fault tolerant. The reason for this is that a neural network is a distributed computing system and is insensitive to partial internal faults. However, without a precise design, it is difficult to guarantee the degree of fault tolerance. Martin and Damper (1993) showed that for multilayered perceptrons, an increase in the number of hidden nodes will not ensure improvement in fault tolerance. Some mechanism to enhance the fault tolerance should be incorporated into the implementation.

In an attempt to enhance the fault tolerant capability, one can create a different architecture or, without modifying the architecture, one can impose additional constraints in the training phase. In this paper, we adopt the latter scheme since it is consistent with the inherent advantage of neural networks.

An optimization problem is usually defined in terms of the minimization of a scalar cost function of a number of variables. If the variables are not constrained by an inequality or equality relationship, optimization is said to be unconstrained. For feedforward networks, the global error E in the weight space is the cost function. E is the mean squared error between the network outputs and the desired training

outputs. In this paper, we say that E is a normal energy term, which is also denoted as E_n, in contrast with another energy term E_c, which will be defined later. We formulate the task of finding a fault tolerant neural network as a constrained optimization problem. The variables (weights) are not constrained by an inequality or equality relationship; instead, the variables are constrained by another cost function, E_c. During training, both E_n and E_c will be minimized.

II. Previous Works on Fault Tolerance

Recently, many researchers have studied the fault tolerant capabilities of the feedforward neural network. The typical methods for studying the feedforward neural network are reviewed below. Chalapathy (1992) formulated the problem of finding a fault tolerant network as a nonlinear constraint optimization problem and solved this problem using a quadratic programming algorithm. The resulting network can guarantee some degree of fault tolerance when any one of the hidden nodes is removed, i.e., permanently stuck-at-0. Minnix (1992) shows that under the condition of a large number of hidden units, i.e., 99 hidden nodes for 5 training patterns trained with noisy input, the network will exhibit some degree of fault tolerance when some hidden nodes fail. The idea is that the effect of adding noise to the input is equivalent to introducing some fault into the network during the training phase. Phatak and Koren (1992) built a fault tolerant neural network by replicating the hidden unit after training was completed. A large amount of redundancy is required to achieve complete fault tolerance even if only one fault is involved. Similar work was done by Phatak and Koren (1995) and Martin and Robert (1993), who called the replicating technique "augmentation".

When considering convergence, the size of the network is usually larger than necessary. If a network has more degrees of freedom than the number of training patterns, then there may exist more than one solution to a specific task, and we may find one solution that can satisfy the fault tolerant requirements. According to this concept, there are some modified training methods which can find the desired fault tolerant solutions. Reed and Sequin (1992) obtained the desired fault tolerance by randomly introducing some faults during training. In their work, one, two or three hidden nodes were selected randomly, and the output was forced to zero or other values, such as +1, −1. When training was completed, the network could tolerate some hidden node stuck-at faults introduced in the training phase. Similar works were done by Huang (1992) and Carlo (1990). However, obtaining the fault tolerance through random selection of faulty hidden neurons is not efficient. This lack of efficiency is due to the fact that with random selection of hidden neurons, the ones which can most influence the fault tolerance capability of the network are not chosen. Bhenam and Amawy (1997) proposed a cyclic selection method to improve the inefficiency. They assume that each neuron in the hidden layer is faulty with a relatively high probability. They think that the most efficient algorithm will be the one in which each hidden layer neuron is assumed to be faulty with probability of one.

To the best of our knowledge, there has been no attempt to take fault tolerance as one of the training goals. Although in the works of Reed and Sequin (1992) and Huang (1992), the fault tolerant capability was achieved through training, they still have no formal definition to explain the mechanism produced by these training methods and can not guarantee the degree of fault tolerance. We believe that this paper is the first work defining a second energy term to be optimized for the purpose of enhancing the fault tolerant property.

III. Notation and Fault Model

Due to the existence of different models of networks and application fields, it is difficult to find a general fault tolerant paradigm which is suitable for all types of neural networks. There are two major groups of networks: feedforward networks and recurrent networks (Nijhuis *et al*., 1990; Peter and Palumbo, 1992). In this paper, we focus our attention on the feedforward neural networks. A general feedforward network consists of an input layer, one or more hidden layers, and an output layer. No feedback or lateral connections are used.

There are two components, nodes and links (weights), which may fail in a feedforward neural network. The frequently encountered type of node failure is the stuck-at fault. Nodes are nonlinear devices, which are fabricated by an operational amplifier. An amplifier may fail in the form of stuck at one of the power supply voltages. For example, it may be stuck-at natural voltage (stuck-at-0), stuck-at maximal voltage or stuck-at minimal voltage. For links, there are two types of failure. The first type of link failure is weight decay; with the implementation of a feedforward network in analog VLSI, the weight values are normally stored as some form of charge storage in capacitance. The entire capacitive storage suffers from the charge leakage effect, and this effect leads to a weight decay phenomenon over time. The second type of link failure is weight disturbance, weights are disturbed by random noise, which occurs with some type of probabilistic distribution.

In this section, we define notations and describe

the fault model adopted throughout this paper. We restrict our attention to the feedforward network with one hidden layer. Such a network can perform arbitrary functions if a sufficient number of hidden nodes is provided.

1. Normal Network

(1) Network $N(w)$ denotes the 3-layered feedforward network. Let $\overline{L}$, $\overline{M}$ and $\overline{Q}$ denote the set of input layer nodes, the set of hidden layer nodes and the set of output layer nodes respectively. The cardinality of sets $\overline{L}$, $\overline{M}$ and $\overline{Q}$ is denoted as L, M and Q, respectively.

(2) Training set $\overline{P}$, is a finite set of ordered pairs. The cardinality of $\overline{P}$ is denoted by P:

$$\overline{P} = \{(\boldsymbol{X}_p, \boldsymbol{Y}_p) | p=1, 2 \ldots, P\}.$$

Here, $\boldsymbol{X}_p$ is the input pattern vector, and $\boldsymbol{Y}_p$ is its corresponding desired output vector:

$$\boldsymbol{X}_p = \{x_p(1), x_p(2), \ldots, x_p(\mathrm{L})\},$$

$$\boldsymbol{Y}_p = \{y_p(1), y_p(2), \ldots, y_p(\mathrm{Q})\}.$$

(3) $\hat{Y}_p(k)$ is the kth component of the actual output vector computed by $N(w)$ for training pattern p.

(4) The error value set $H=\{e_p(q) | p=1, 2, \ldots, P\ q=1, 2, \ldots, Q\}$, where

$$e_p(q) = |y_p(q) - \hat{y}_p(q)| \quad \textbf{(1)}$$

so there are in total $P \times Q$ possible error values in set H. The maximal value in set H is denoted by $Max(H)$, and $Sum(H)$ denotes the total sum of all elements in H.

(5) The normal energy E_n, is the sum of the squared difference between the desired outputs and actual outputs over all of the output units of all training patterns. The most popular training method, BP based on the Generalized Delta Rule, is used to minimize the value of this function:

$$E_n = \frac{1}{2} \sum_{p=1}^{P} \sum_{q=1}^{Q} [y_p(q) - \hat{y}_p(q)]^2 . \quad \textbf{(2)}$$

The goal of training is to find am exact set of weights such that the global error E_n is the smallest in the whole error space. In practice, since the value of the global minimum is unknown, a small global error is considered acceptable. Thus, for normal BP training, the convergence criterion is that E_n is small enough. However, as pointed out by Arun and Polycarpou (1997), it is always possible to have a network with a small global error E_n but with a high $Max(H)$ value. In order to ensure that the normal network and the network with a failure node can function normally, we change the convergent criterion from $E_n < \sigma$ to $Max(H) < \alpha$.

2. Network with Failure Nodes

(1) $\overline{m}_f$ is the subset of the hidden node set $\overline{M}$, where the cardinality of $\overline{m}_f$ is denoted by m, and the subscript f denotes the type of fault. Thus, there are m nodes in set $\overline{m}_f$, and these nodes are all stuck-at faulty f. Given cardinalities M and m of the set $\overline{M}$ and $\overline{m}_f$, there exist

$$R = \binom{M}{m} \quad \textbf{(3)}$$

possible different subsets $\overline{m}_f$ of $\overline{M}$.

(2) $N^{\overline{m}_f}(w)$ derived from a normal network $N(w)$ with some hidden nodes in set $\overline{m}_f$.

(3) $\hat{y}_p^{\overline{m}_f}(k)$ denotes the kth component of the actual output vector computed by $N^{\overline{m}_f}(w)$ for training pattern p.

(4) The error value set

$$H^{\overline{m}_f} = \{e_p(q, \overline{m}_f) | p=1, \ldots, P;\ q=1, \ldots, Q;\ \forall \overline{m}_f \in \overline{M}\}, \quad \textbf{(4)}$$

where

$$e_p(q, \overline{m}_f) = \left| \hat{y}_p(q) - \hat{y}_p^{\overline{m}_f}(q) \right| . \quad \textbf{(5)}$$

The cardinality of set $H^{\overline{m}_f}$ is

$$\left| H^{\overline{m}_f} \right| = P \times Q \times R . \quad \textbf{(6)}$$

(5) The constraint energy term E_c, measures the difference between the normal network and the network with failure nodes:

$$E_c^{\overline{m}_f} = \frac{1}{2} \sum_{p=1}^{P} \sum_{q=1}^{Q} [\hat{y}_p(q) - \hat{y}_p^{\overline{m}_f}(q)]^2 . \quad \textbf{(7)}$$

IV. Constraint Backpropagation (CBP) Algorithms

In this section, we describe the CBP training algorithm. According to different types of node functions used, different numbers of hidden node failures tolerated and different types of faults, there is a set of CBP training algorithms CBP1 through CBP6. Table 1 lists the various features among them.

Table 1. Various Types of Fault Tolerant Constraint Training

ALG.	Node Function	Number of Faulty Nodes	Type of Fault
CBP1	Sigmoid	1	Stuck-at-0
CBP2	Sigmoid	1	Stuck-at-1
CBP3	Sigmoid	1	Stuck-at-0,1
CBP4	Sigmoid	2	Stuck-at-0
CBP5	Linear	1	Stuck-at-0
CBP6	Linear	2	Stuck-at-0

1. choose α and β such that $\alpha + \beta < 0.5$
   limit_epoch=10000
   epoch=0
   $m = 1$
   f=0
2. **loop** 1: for each training pattern p
   propagate forward for $N(w)$
   if $\underset{q\in Q}{Max}\ e_p(q) > \alpha$ **then** adjust weights-
   loop 2: for every subset $\overline{m}_f$ of hidden nodes
   propagate forward for $N^{\overline{m}_f}(w)$
   compute $e_p(q,\overline{m}_f),\ q \in Q$
   end-of-loop 2
   find $\overline{z}_f \in \overline{M}$ such that $e_p(q,\overline{z}_f) = \underset{\overline{m}_f \in \overline{M}}{Max} e_p(q,\overline{m}_f)$
   if $e_p(q,\overline{z}_f) \geq \beta$ **then**
   { propagate-forward for $N^{\overline{z}_f}(w)$
   adjust weights
   }
   end-of-loop 1
3. *epoch*++
4. **if** ($Max(H) < \alpha$ **and** $Max(H^{\overline{m}_f}) < \beta$) **or**
   (*epoch*>*limit_epoch*)
   then finish the training
5. goto step 2

Fig. 1. Constraint training algorithm 1 (CBP 1).

CBP1 is our original work – the tolerance of a single stuck-at-0 fault in a hidden layer. This type of fault tolerance is equivalent to a network that functions well when any one of the hidden nodes is removed. This type of fault was also discussed by Phatak and Koren (1992) as mentioned in Section II. Figure 1 is the CBP1 training algorithm. CBP2 and CBP5 are similar to CBP1 except for the stuck-at-1 fault and the linear activation functions.

A network trained with CBP3 (Fig. 2) can tolerate one hidden node failure, which may be stuck-at-1 or stuck-at-0. Any single hidden node stuck-at-1 or stuck-at-0 will not degrade the performance of the network. During training, the weights are adjusted 3 times for one pattern passed: the first adjustment reduces the global error, the second reduces the maximal stuck-at-0 fault and the last reduces the maximal stuck-at-1 fault.

Figure 3 shows the CBP4 training algorithm. A network trained with CBP4 can tolerate one or two hidden nodes stuck-at-0. Removing one or two hidden nodes from the network will not degrade the performance of the network. During training, the weights are adjusted three times for one pattern passed: the first adjustment is to reduces the global error, the second reduces the maximal stuck-at-0 fault for one node failure, and the last also reduces the maximal stuck-at-0 fault with the exception of the case in which two nodes fail.

1.chose α and β such that $\alpha + \beta < 0.5$
   limit_epoch=10000
   epoch=0
   $m = 1$
2. **loop** 1 :for each training pattern p
   propagate forward
   if $\underset{q\in Q}{Max}\ e_p(q) \geq \alpha$ then adjust weights
   f=0
   loop 2: for every subset $\overline{m}_f$ of hidden nodes
   propagate forward for $N^{\overline{m}_f}(w)$
   compute $e_p(q,\overline{m}_f)\quad q \in Q$
   end-of-loop 2
   find $\overline{z}_f \in \overline{M}$ such that $e_p(q,\overline{z}_f) = \underset{\overline{m}_f \in \overline{M}}{Max} e_p(q,\overline{m}_f)$
   if $e_p(q,\overline{z}_f) \geq \beta$ **then**
   { propagate forward for $N^{\overline{z}_f}(w)$
   adjust weights
   }
   f=1
   loop 3 : for every subset $\overline{m}_f$ of hidden node set
   propagate forward for $N^{\overline{m}_f}(w)$
   compute $e_p(q,\overline{m}_f)\quad q \in Q$
   end-of-loop 3
   find $\overline{z}_f \in \overline{M}$ such that $e_p(q,\overline{z}_f) = \underset{\overline{m}_f \in \overline{M}}{Max} e_p(q,\overline{m}_f)$
   if $e_p(q,\overline{z}_f) \geq \beta$ **then**
   { propagate forward for $N^{\overline{z}_f}(w)$
   adjust weights
   }
   end-of-loop 1
2. *epoch*++
3. **if** ($Max(H) < \alpha$ **and** $Max(H^{\overline{m}_0}) < \beta$ **and** $Max(H^{\overline{m}_1}) < \beta$)
   or (*epoch*>*limit_epoch*)
   then finish the training
5. goto step 2

Fig. 2. Constraint training algorithm 3 (CBP 3).

1.chose α and β such that $\alpha+\beta<0.5$
limit_epoch=10000
epoch=0
f=0
2. **loop** 1 :for each training pattern p
propagate forward
if $\max_{q\in Q} e_p(q)\geq\alpha$ then adjust weights
f=0
$m=1$
loop 2: for every subset $\overline{m}_f$ of hidden node set
propagate forward for $N^{\overline{m}_f}(w)$
compute $e_p(q,\overline{m}_f)$ $q\in Q$
end-of-loop 2
find $\overline{z}_f\in\overline{M}$ such that $e_p(q,\overline{z}_f)=\max_{\overline{m}_f\in\overline{M}} e_p(q,\overline{m}_f)$
if $e_p(q,\overline{z}_f)\geq\beta$ **then**
{ propagate forward for $N^{\overline{z}_f}(w)$
adjust weights
}
$m=2$
loop 3 : for every subset $\overline{m}_f$ of hidden node
propagate forward for $N^{\overline{m}_f}(w)$
compute $e_p(q,\overline{m}_f)$ $q\in Q$
end-of-loop 3
find $\overline{z}_f\in\overline{M}$ such that $e_p(q,\overline{z}_f)=\max_{\overline{m}_f\in\overline{M}} e_p(q,\overline{m}_f)$
if $e_p(q,\overline{z}_f)\geq\beta$ **then**
{ propagate forward for $N^{\overline{z}_f}(w)$
adjust weights
}
end-of-loop 1
3. *epoch*++
4. **if** $(Max(H)<\alpha$ **and** $Max(H^{\overline{m}_0})|_{m=1}<\beta$ **and** $Max(H^{\overline{m}_0})|_{m=2}<\beta)$
or (*epoch*>*limit_epoch*)
then finish the training
5. goto step 2

Fig. 3. Constraint training algorithm 4 (CBP 4).

V. Experimental Results

1. Fault Tolerant Capability

CBP training has been tested on various problems, such as speech recognition, character recognition, parity problems, etc. For clarity, we use recognition of 26 English characters. The training samples were 26 English characters arranged in a 7×6 pixel array, and the outputs were the corresponding ASCII code (Fig. 4). In this problem, we have 42 input nodes and 8 output nodes. Currently, no formal method is available for predicting the absolute minimal number of hidden nodes needed to solve a particular problem. It is expected that most neural networks will have some redundant nodes in the hidden layer. These redundant nodes might not be of any help in improving the fault tolerance capability, as every node must be included in the computation, so none of them can be removed from the network.

Huang (1992) has proved that the least upper bound of the number of hidden nodes needed to solve a problem equals the number of training patterns minus one. To tolerate one or two hidden nodes stuck at fault, the number of hidden nodes chosen must be equal to the number of training patterns. We use symbols N1 through N44 to denote networks trained with various constraint training methods and with various degrees of fault tolerance. The fault tolerance performance was evaluated using two values; *Max*(*H*) and *Sum*(*H*). The lower the values, the better the fault tolerant capability the networks have. In the case where *Max*(*H*) is equal to zero, a network with node failures will perform in exactly the same way function as a normal network. Tables 2-7 show the results.

All training patterns passed once through the network is called an epoch, which is often taken in most of the literature as a measurement of convergent speed. Since there is more than one adjustment of weights for a pattern passed in CBP training, one epoch alone can not represent the convergent speed precisely. Tables 2-7 also show the number of computing cycles in training. A computing cycle here is defined as a single adjustment of all weights.

2. Generalization Capability

One important performance measure of a feedforward network is its generalization. Generalization evaluates the correctness ratio for novel patterns. The generalization error decreases in an early period of training, reaches a minimum and then increases as training goes on while the training error monotonically decreases. A network that has a high correctness ratio for training patterns but a poor ratio for novel patterns does not function well. In this sub-section, we explore the generalization capability of networks trained by CBP and compare the results with those of networks trained by normal BP. We generated four test sets from the training sets, the Hamming distance between the training sets and test sets are one, two, three and four, respectively. Each test set contained 156 test patterns. The values in column three to six of Table 8 are the correctness ratios for recognition of the test sets. In

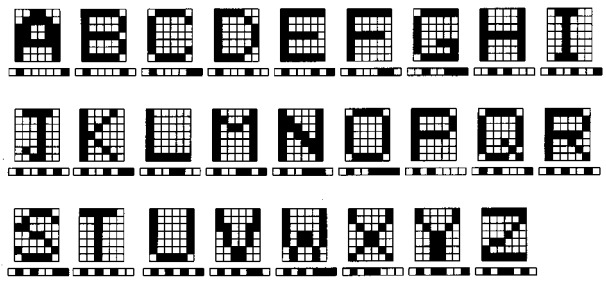

Fig. 4. Bitmap of 26 English characters and the corresponding ASCII code.

Table 2. Fault Tolerant Performance of CBP1 with α=0.1

Network	Training ALG.	β	Epochs	Computing Cycles	Stuck-at-0 Fault (m=1, f=0)	
					$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$
N1	BP	X	612	612	0.81	97.15
N2	CBP1	0.40	661	1322	0.377	40.09
N3	CBP1	0.35	666	1332	0.348	39.99
N4	CBP1	0.30	675	1350	0.277	35.11
N5	CBP1	0.25	693	1386	0.227	33.83
N6	CBP1	0.20	714	1428	0.199	29.85
N7	CBP1	0.15	759	1518	0.147	25.47
N8	CBP1	0.10	957	1914	0.099	19.18
N9	CBP1	0.05	1300	2600	0.049	10.90

Table 3. Fault Tolerant Performance of CBP2 with α=0.1

Network	Training ALG.	β	Epochs	Computing Cycles	Stuck-at-1 Fault (m=1, f=1)	
					$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$
N1	BP	X	612	612	0.78	191.87
N10	CBP2	0.40	693	1386	0.384	85.84
N11	CBP2	0.35	700	1400	0.348	75.45
N12	CBP2	0.30	707	1414	0.299	77.29
N13	CBP2	0.25	756	1512	0.249	69.32
N14	CBP2	0.20	777	1554	0.189	61.23
N15	CBP2	0.15	1100	2200	0.147	52.42
N16	CBP2	0.10	1212	2424	0.099	38.90
N17	CBP2	0.05	1343	2686	0.049	21.30

Table 4. Fault Tolerant Performance of CBP3 with α=0.1

Network	Training ALG.	β	Epochs	Computing Cycles	Stuck-at-0 Fault (m=1, f=0)		Stuck-at-1 Fault (m=1, f=1)	
					$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$	$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$
N1	BP	X	612	612	0.81	97.15	0.78	191.87
N18	CBP3	0.40	723	2169	0.336	28.71	0.341	63.96
N19	CBP3	0.35	723	2169	0.336	28.71	0.341	63.96
N20	CBP3	0.30	730	2190	0.282	25.93	0.293	53.73
N21	CBP3	0.25	740	2220	0.243	23.59	0.248	48.29
N22	CBP3	0.20	749	2247	0.193	23.82	0.195	47.2
N23	CBP3	0.15	776	2328	0.146	20.94	0.148	39.83
N24	CBP3	0.10	855	2565	0.098	16.18	0.099	29.11
N25	CBP3	0.05	1121	3363	0.049	9.03	0.049	15.46

general, networks that possess better fault tolerant capability also possess better generalization. Similar results can also be obtained for a network trained by other versions of the CBP training method.

3. Mechanism of CBP Training Algorithm

In an attempt to understand why the network trained with the CBP training method exhibited the fault tolerant property, we performed further analysis on networks trained with CBP, then compared these results with those networks trained with normal BP and tried to find the differences between them. We first discuss networks with sigmoid function. Figure 5 shows the mean absolute net input of the output layer node for N2 through N33. Examining Fig. 5, we see that for networks with a greater degree of fault tolerance, the mean of the absolute net input tends to be large.

Table 5. Fault Tolerant Performance of CBP4 with α=0.1

Network	Training ALG.	β	Epochs	Computing Cycles	Stuck-at-0 Fault (m=1, f=0)		Stuck-at-0 fault (m=2, f=0)	
					$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$	$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$
N1	BP	X	612	612	0.81	97.15	0.95	2258.60
N26	CBP4	0.40	759	2277	0.16	6.02	0.384	201.84
N27	CBP4	0.35	760	2280	0.16	4.75	0.343	166.33
N28	CBP4	0.30	819	2457	0.14	3.15	0.285	118.96
N29	CBP4	0.25	839	2517	0.11	2.75	0.249	107.81
N30	CBP4	0.20	879	2637	0.08	1.89	0.195	81.71
N31	CBP4	0.15	1181	3543	0.04	1.81	0.148	80.23
N32	CBP4	0.10	2676	8028	0.04	1.79	0.095	78.24
N33	CBP4	0.05	4134	12429	0.04	1.78	0.047	74.28

Table 6. Fault Tolerant Performance of CBP5 with α=0.1

Network	Training ALG.	β	Epochs	Computing Cycles	Stuck-at-1 Fault (m=1, f=1)	
					$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$
N34	BP	X	784	784	0.562	258.37
N35	CBP5	0.40	817	1634	0.392	251.88
N36	CBP5	0.35	917	1834	0.341	252.00
N37	CBP5	0.30	980	1960	0.292	241.76
N38	CBP5	0.25	1061	2122	0.241	232.65
N39	CBP5	0.20	1306	2612	0.199	214.99
N40	CBP5	0.15	X	X	–	–

Table 7. Fault Tolerant Performance of CBP6 with α=0.1

Network	Training ALG.	β	Epochs	Computing Cycles	Stuck-at-0 Fault (m=1, f=0)		Stuck-at-0 Fault (m=2, f=0)	
					$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$	$Max(H^{\overline{m}_f})$	$Sum(H^{\overline{m}_f})$
N34	BP	X	784	784	0.562	258.37	0.563	4339.89
N42	CBP6	0.40	823	1646	0.294	220.96	0.399	4424.42
N43	CBP6	0.35	918	2754	0.263	216.83	0.348	4339.23
N44	CBP6	0.30	1204	3612	0.223	213.82	0.299	4237.03
N45	CBP6	0.25	X	X	–	–	–	–

Nodes with large net input are said to be more saturated when a sigmoid node function is used. Figure 6 shows the mapping graph of the sigmoid function. The horizontal axis corresponds to the net input values of a node, and the vertical axis corresponds to the node's output values. When an output node has an absolute net input value larger than or equal to 7, we say that this node is in a state of saturation. In a saturated state, a minor change in net input will not affect the node's output. The other state is the sensitive state, as any changes in net input will lead to output changes. Therefore, the better fault tolerant property can be attributed to output layer nodes which tend to be saturated.

Since networks with linear function nodes always stay in the sensitive region, the fault tolerant capability is worse than that of networks with sigmoid function nodes. Compare Table 2 with Table 6, we find that the best $Max(H^{\overline{m}_f})$ (m=1, f=0) is 0.049 for CBP1 but 0.199 for CBP5. Similar results can be seen in Table 4 and Table 6; the best $Max(H^{\overline{m}_f})$ (m=2, f=0) is 0.047 for CBP4 but 0.299 for CBP6.

4. Output Node Saturation

We have also developed a technique called output nodes saturation (ONS) for networks with sigmoid function nodes. Combining CBP and ONS, we obtain

Table 8. A Comparison of Generalization Capability of the Networks Trained by BP and CBP1

Network	Training method	Hamming distance				Average
		1	2	3	4	
N1	BP	92.95	76.28	60.90	42.95	68.27
N2	CBP1	96.78	78.85	71.79	48.08	73.88
N3	CBP1	96.78	78.21	70.51	48.08	73.40
N4	CBP1	96.15	80.13	70.51	48.08	73.72
N5	CBP1	96.15	80.27	71.79	46.79	73.75
N6	CBP1	95.51	80.13	71.79	50.64	74.51
N7	CBP1	97.87	82.69	71.79	53.21	75.64
N8	CBP1	95.51	83.33	73.08	54.49	76.67
N9	CBP1	95.51	83.33	72.44	53.85	76.28

a complete fault tolerant network. Complete fault tolerant networks possess an attractive feature in that when faults occur, the networks perform in exactly the same way as normal networks. On the other hand, in a complete fault tolerant network, $Max(H^{\overline{m}_f})$ is equal to zero.

In normal applications, for a desired output of '1', the network must be trained to have an output larger than 0.5 (positive net input) and it must have an output value smaller than 0.5 (negative net input) for the desired output '0'. In all of our experiments, the training error was less than 0.1, and under the condition of hidden node failure, the degradation of output was within the range of 0.05 to 0.4. Thus, for a desired output of '1', a network with failing hidden nodes had an output larger than 0.5 and had an output less than 0.5 for a desired output of '0'. We then pushed the output nodes from the sensitive region into the saturation region by multiplying a constant C by the weights between the hidden layer and the output layer. If the constant was large enough, both the normal network and network with failure had an output of '1' for a desired output of '1', and '0' for a desired output of '0'. Figure 7 shows the $Max(H)$ and the $Max(H^{\overline{m}_f})$ values after multiplying various values of the constant C of network N2 trained by CBP1. From Fig. 7, we see that when we multiplied a constant larger than 11 by the weights between the hidden layer and output layer, network N2 became a complete fault tolerant network. Similar results can be obtained for other networks.

5. Convergent Property of CBP Training Algorithm

In the process of CBP training, there are multiple adjustments of weights for each training pattern passed. For example, at the n+1 iteration of CBP3, weight adjustment is performed three times.

$\nabla'(w)$ is added to minimize the global error function E_n, and $\nabla''(w)$ and $\nabla'''(w)$ are added to minimize the constraint energy functions $E_c^{\overline{m}_0}$ and $E_c^{\overline{m}_1}$. A contradiction may occurr; i.e., when $\nabla'(w)$ is added to the weights, E_n decreases but the other two constraint energies increase. Table 9 shows the worst contradiction case and ideal case. From the results of our experiments, in the first 30% of CBP training, contradiction situation appears, then a compromise situation between the contradiction and the ideal situation appears, and in the last stage, the ideal situation appears.

VI. Conclusion

We have proposed a training method in which the task of finding a fault tolerant neural network is treated as a constraint optimization problem. This problem is solved by using the constraint BP training method. During training, both the normal energy and constraint energy are minimized by means of multiple weight

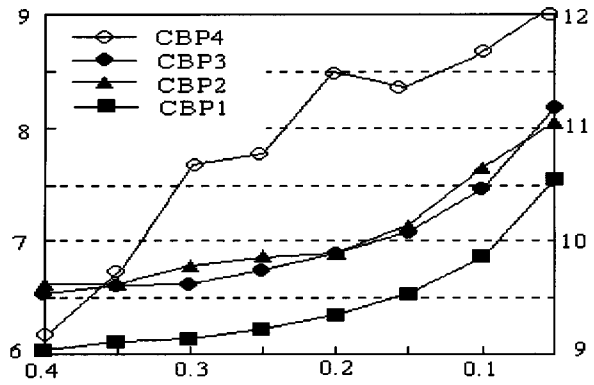

Fig. 5. The horizontal axes are the β values of the networks. The vertical axes are the mean values of absolute net input of output layer nodes, the left hand side axis represents the scales for CBP1 to CBP3, and the right hand side represents the scales for CBP4.

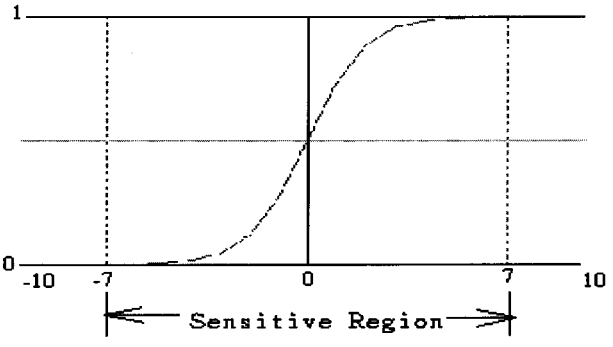

Fig. 6. Sigmoid function, beyond the sensitivity region is the saturation region.

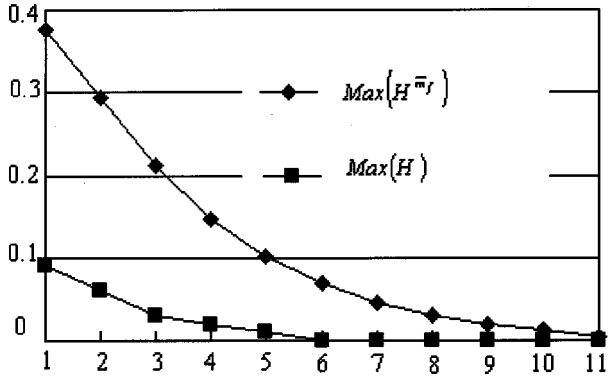

Fig. 7. The $Max(H^{\overline{m}_I})$ and the $Max(H)$ of network N2 after multiplying various values of constant C. The horizontal axis represent the various values of the constant multiplied by weights between the hidden layer and output layer.

Table 9. Tendency of the Energy Function

modification of weights	contradiction case, m=1			ideal case, m=1		
	E_n	$E_c^{\overline{m}_0}$	$E_c^{\overline{m}_1}$	E_n	$E_c^{\overline{m}_0}$	$E_c^{\overline{m}_1}$
$\nabla'(w)$	↓	↑	↑	↓	↓	↓
$\nabla''(w)$	↑	↓	↑	↓	↓	↓
$\nabla'''(w)$	↑	↑	↓	↓	↓	↓

Notes: (1) $w'(n+1)=w(n)+\nabla'(w)$
(2) $w''(n+1)=w'(n+1)+\nabla''(w)$
(3) $w(n+1)=w''(n+1)+\nabla'''(w)$

adjustments. The constraint function should be well defined; otherwise, the network may never converge.

In the traditional digital computing system, any internal fault may cause the system to break down. Neural networks exhibit graceful performance degradation under partial internal faults. However, graceful performance degradation does not guarantee availability, especially for critical tasks. The CBP training algorithm proposed in this paper provides some degree of fault tolerance under hidden node failure. Furthermore, by combining the CBP training algorithm and the output node saturation strategy, we can construct a fault free network, which has exactly the same performance as a normal network when some hidden nodes fail.

The output layer is the most critical part of a feedforward network, any failure in it will impair system performance. To overcome the problem of output node failure, one might use the TMR design as mentioned in the work of Khunasaraphan *et al*. (1994) or use the robust neuron suggested by Chu and Benjamin (1990).

CBP requires extra computation time at each iteration in deciding which hidden node will cause the maximal error, and in then adjusting the weights again according to this finding. The convergent property will not suffer due to extra adjustment of weights because the constraint functions are well defined. Other types of constraint functions which enable the network to possess other good properties, such as generalization, sensitivity, etc., are still being studied.

References

Abhijit, S. P. and R. B. Macy (1996) *Pattern Recognition with Neural Networks in C*$^{++}$. CRC Press Inc., Kansas City, KS, U.S.A.

Arun, T. and M. Polycarpou (1997) Neural network-based robust fault diagnosis in robotic system. *IEEE Trans. on Neural Networks*, **8**(6), 1410-1420.

Bhenam, S. A. and A. E. Amawy (1997) on fault tolerant trarning of feedforward neural networks. *Neural Networks*, **10**(3), 539-553.

Carlo, H. S. (1990) Fault tolerance in artificial neural networks. International Joint Conference on Neural Networks, San Diego, CA, U.S.A.

Chalapathy, N. (1992) Maximally fault tolerant neural networks. *IEEE Trans. on Neural Networks*, **3**(1), 16-22.

Chu, L. C. and W. W. Benjamin (1990) Fault tolerant and neural networks with hybrid redundancy. International Joint Conference on Neural Networks, San Diego, CA, U.S.A.

Diamantaras, K. l. and S. Y. Kung (1996) *Principle Component Neural Networks Theory and Application*. John Wiely and Sons, Inc., New York, NY, U.S.A.

Huang, S. C. (1992) Bound on the number of hidden neurons in multi layer perceptrons. *IEEE Trans. on Neural Networks*, **3**(1), 47-55.

Khunasaraphan, C., K. Vanapipat, and C. Lursinsap (1994) Weight shifting techniques for self-recovery neural networks. *IEEE Trans. on Neural Networks*, **5**(4), 651-658.

Martin, D. E. and R. 1. Damper (1993) Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. *IEEE Trans. On Neural Networks*, **4**(5), 788-793.

Minnix, J. I. (1992) Fault tolerance of the backpropagation neural network trained on noisy inputs. International Joint Conference on Neural Networks, Baltimore, MD, U.S.A.

Nijhuis, J., B. Hofflinger, A. Schaik, and L. Spaanenburg (1990) Limit to the fault tolerance of a feedforward neural network with learning. International Joint Conference on Neural Networks, San Diego, CA, U.S.A.

Peter, W. P. and D. L. Palumbo (1992) Performance and fault-tolerance of neural networks for optimization. *IEEE Trans. on Neural Networks*, **4**(4), 600-614.

Phatak, D. S. and I. Koren (1992) Fault tolerance of feedforward neural net for classification tasks. International Joint Conference on Neural Networks, Baltimore, MD, U.S.A.

Reed, D. C. and C. H. Sequin (1992) Fault tolerance training improves generalization and robustness. International Joint Conference on Neural Networks, Baltimore, MD, U.S.A.

前饋式神經網路的容錯訓練

佘步雲* 謝文雄**

*國立中山大學電機工程研究所
**國立中山大學資訊工程研究所

摘 要

容錯性能是神經網路數個效能度量中重要的一個，且常被認為是神經網路固有的一項特點，但若未經精確的設計，吾人並不能保證容錯的程度。本研究在檢視前饋式神經網路的容錯性質並提出一個容錯訓練方法，經此方法訓練的神經網路，在一或二個隱藏層節點毀損時仍能保有相當程度的容錯能力。為達到容錯的目的，吾人定義一新的能量函數稱作限制形能量，在訓練的過程當中，正規能量和限制形能量都將逐漸降低。將此容錯式訓練輔以本研究另提出之輸出節點飽和法，吾人可訓練出在一些隱藏節點故障時和正常網路保有完全一樣效能的神經網路。實驗結果同時也證實，經由容錯訓練的神經網路有較佳的歸納性質。