PAPER

Dynamic Constructive Fault Tolerant Algorithm for Feedforward Neural Networks

Nait Charif HAMMADI†, Toshiaki OHMAMEUDA††, Keiichi KANEKO††, *and* Hideo ITO††, *Members*

SUMMARY In this paper, a dynamic constructive algorithm for fault tolerant feedforward neural network, called DCFTA, is proposed. The algorithm starts with a network with single hidden neuron, and a new hidden unit is added dynamically to the network whenever it fails to converge. Before inserting the new hidden neuron into the network, only the weights connecting the new hidden neuron to the other neurons are trained (i.e., updated) until there is no significant reduction of the output error. To generate a fault tolerant network, the relevance of each synaptic weight is estimated in each cycle, and only the weights which have their relevance less than a specified threshold are updated in that cycle. The loss of a connections between neurons (which are equivalent to stuck-at-0 faults) are assumed. The simulation results indicate that the network constructed by DCFTA has a significant fault tolerance ability.

key words: *feedforward neural network, dynamic constructive algorithm, fault tolerance, DCFTA*

1. Introduction

Multilayer Neural Networks (NNs) are now widely used in pattern recognition/classification applications. It has been shown that feedforward networks are capable of implementing any input-output mapping provided that they have a sufficient number of hidden neurons. Nevertheless, the performance of the network depends on many arbitrary chosen parameters like learning algorithm, weights initialization, the number of hidden neurons (i.e., network's size), and the activation function of neurons [1].

Typically, NNs training schemes require network size to be set before learning is initiated [2]. A fundamental question that raises when talking about fault tolerance ability of feedforward neural networks is , how many hidden neurons are necessary to be fault tolerant. A number of researchers have focused their attention either on the optimization of the number of hidden neurons necessary for the NN to learn a specified task, or on the fault tolerance enhancement of NNs with a fixed number of hidden neurons.

So far, mainly fixed architectures with a fixed number of hidden neurons have been considered for feedforward neural network. One of the few exceptions is the cascade architecture proposed by Fahlman [3], [4] and its versions [5], [6]. However the cascade correlation proposed by Fahlman is not a simple multilayer network since each new neuron is itself a new layer, and the number of layer is equal to the number of hidden neurons, thus, in case of a large network, the time necessary for an input to propagate to the output may be critical. One of the characteristics of conventional single hidden layer network is its parallel processing architecture (i.e., the neurons in a given layer can process in parallel), thus the delay is minimal and it can be used in the applications that require a quick response.

Although fault tolerance is frequently cited as an important property of NNs [7], the loss of single weight is frequently sufficient to completely disrupt a learned function. The backpropagation learning algorithms do not make optimal use of redundant resources. Recently extensive research has proved that NNs are not intrinsically fault tolerant, and the fault tolerance has to be enhanced by adequate scheme [8].

A number of methods have been proposed to enhance the fault tolerance ability of NNs. In [9], A.F. Murray et al. analyzed the effect of analog noise injection on the synaptic weights during multilayer neural network training on the fault tolerance property. Procedures to build fault tolerant NNs by replicating the hidden units are presented [8], [10], and the minimum redundancy required to tolerate all possible single faults is analytically derived [10]. Using error correcting code, a fault tolerant design which can correct an error at the output layer neuron was presented [11]. A learning algorithm for fault tolerant NNs is proposed in [12].

However, a constructive algorithm that incorporates a mechanism that makes the constructed network more fault tolerant has not been proposed yet. In this paper, assuming that a physically plausible type of fault is the loss of a connection between two neurons, we propose a dynamic constructive algorithm for the feedforward network called DCFTA. This algorithm estimates, in each learning cycle, the relevance/sensitivity of each weight to the output error, and updates only the weights which have their relevance less than a specified threshold. And whenever the learning process stagnates, a new hidden neuron is added to the network.

This paper is organized as follows. First, in Sect. 2 we present a dynamic constructive learning algorithm for feedforward neural network. In Sect. 3, we incor-

Manuscript received May 29, 1997.

†The author is with the Graduate School of Science and Technology, Chiba University, Chiba-shi, 263 Japan.

††The authors are with the Faculty of Engineering, Chiba University, Chiba-shi, 263 Japan.

porate in the constructive algorithm a mechanism that produces a fault tolerant network. The simulation results that evaluate the proposed algorithm are presented and discussed in Sect. 4, in this section we compare the proposed technique to the method of M.D. Emmerson et al. called *augmentation*[7].

2. Dynamic Constructive Algorithm: DCA

In this section a dynamic constructive algorithm (denoted by DCA) is presented. The aim is to generate NNs with nearly minimum number of hidden neurons. DCA is extended to be fault tolerant in the next section. Since the number of inputs and the number of outputs are fixed by the problem and the input/output representation that the designer has chosen, they are also fixed in this paper.

It is intended to construct dynamically a NN where the input layer is fully connected to the hidden layer which is also fully connected to the output layer. There are also a bias input and a bias hidden, permanently set at 1.

The output o_j of the jth neuron is given by

$$o_j = f\left(\sum_{i=0}^{K} w_{ij} o_i\right), \tag{1}$$

where w_{ij} is the synaptic weight corresponding to the connection from the ith neuron in the previous layer to the jth neuron, o_i is the output of the ith neuron, K is the number of neurons that feeds the jth neuron (which is equal to the number of neurons in the previous layer), the bias w_{0j} is treated as a synaptic weight connected to a fixed input $o_0 = 1$, and f is the sigmoid activation function given by

$$f(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

The hidden neurons are added dynamically one by one in DCA. Each new hidden neuron receives a connection from each of the network's inputs. In the opposite of the cascade correlation algorithm[3], all the *input-to-hidden* and *hidden-to-output* weights are trained repeatedly, not only the *hidden-to-output* weights.

The DCA consists of cyclic repetition of three phases, *Train-Normal-Net* (denoted by P_{TNN}), *Train-Candidates* (P_{TC}), and *Neurons-Addition* (P_{NA}), after *Initialization* phase. The phases are explained in detail later.

The *Initialization* phase initializes the parameters such as the learning rate and the values of the weights. P_{TNN} starts with a single hidden neuron and all the weights are trained with the backpropagation algorithm which minimizes the mean-squared error (objec-

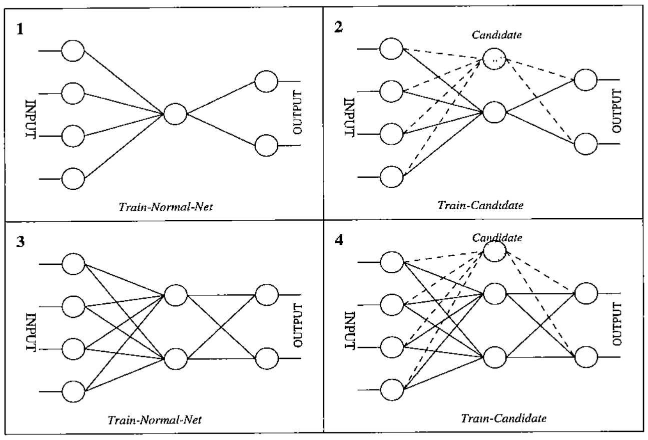


Fig. 1 The network architecture. A candidate neuron is created temporarily and the candidate weights on the weights on the dashed lines are trained while all the others are frozen.

tive function) given bellow

$$E = \frac{1}{2P}\sum_{p=1}^{P}\sum_{k=0}^{K}(t_k^p - o_k^p)^2, \tag{3}$$

where P is the number of patterns in the training set, K is the number of neurons in the output layer, and t_k^p is the target output and o_k^p practical output of the kth neuron for the pth pattern. One training cycle corresponds to the presentation of all the patterns in the training set to the network just once. When no significant error reduction has occurred after a given number of cycles T_N, we test, if the convergence criteria is satisfied the DCA stops; otherwise there must be a residual error that should be reduced. To achieve this, P_{TC} starts, and an independent neuron (a *candidate*) is created (Fig. 1). This neuron is connected to all the input neurons and all output neurons (it behaves like a hidden neuron) and only *input-to-candidate* and *candidate-to-output* weights are trained to minimize the output error. All the previously trained weights are temporary "frozen." P_{TC} stops when there is no significant reduction of the output error after T_{cand} cycle. Then P_{NA} starts, and the candidate neuron is definitely added to the network as a normal hidden neuron. And the whole network is then trained in P_{TNN}. This process is repeated until the convergence criteria is satisfied or the maximum network size is reached.

In P_{TC}, it is possible to train all the weights including the previously trained weights. However, since the candidate weights are set randomly, it is assumed that they should be brought to nearly the same level of trainability as the previously trained weights before being definitely added to the network as a normal weights.

Figure 1 presents an example where a network with four input neurons and two output neurons is being dynamically constructed. In the phase P_{TNN} (1 and 3) all the weight are updated, in the phase P_{TC} only the weights on the dashed connections are updated.

The advantage of DCA is that it can automatically find the size of the NN without specifying it before the training begins.

Instead of a single candidate neuron in P_{TC}, it is generalized to train a pool of candidate neurons as in Fehlman cascade-correlation algorithm, so that the generalized DCA can select the best neuron among the pool after the P_{TC} training phase. Each candidate with different set of initial weights, is temporarily connected to the output of every input neuron, and its output is also temporarily connected to every neurons in a virtual output layer, where a virtual output layer is a temporal layer of the same size as the original output layer (i.e. they have the same number of neurons). Figure 2 shows an example of two candidate neurons with the corresponding virtual output layers. The output v_k^p of the kth neuron of the virtual layer is given by

$$v_k^p = f(rest_k^p + w_{ck}^c o_c^p), \tag{4}$$

where w_{ck}^c is the weight corresponding to the connection between the candidate neuron c and kth neuron of the virtual output layer, o_c^p is the output of the candidate neuron, and $rest_k^p$ is a resulting input to the kth neuron from the actual network when the pth pattern is presented to the network. $rest_k^p$ is given by

$$rest_k^p = \sum_{h=0}^{H} w_{hk} o_h^p, \tag{5}$$

where H is the number of hidden neurons in this stage of the learning process, w_{hk} the weight on the connection from the hth hidden neuron and kth output neuron, and o_h^p is the output of the hth hidden neuron. The output o_c^p of a candidate neuron is give by

$$o_c^p = f\left(\sum_{i=0}^{N} w_{ic} x_i^p\right), \tag{6}$$

where w_{ic} is the weight on the connection from the ith neuron of the input layer to the candidate, x_i is the ith element of the input vector, and N is the number of neurons in the input layer.

In the P_{TC} (*Train-Candidate*) phase, for each candidate neuron, the *input-to-candidate* and *candidate-to-virtual* weights are trained to minimize the output error given by

$$E_{cand} = \frac{1}{2P} \sum_{p=1}^{P} \sum_{k=0}^{K} (t_k^p - v_k^p)^2. \tag{7}$$

Each candidate is trained until there is no significant reduction in its corresponding error E_{cand}, after T_{cand} cycles. The P_{TC} phase stops when there is no significant reduction in any E_{cand}. Among the candidate neurons, the one with the minimum output error E_{cand} becomes a new hidden neuron and all the other hidden

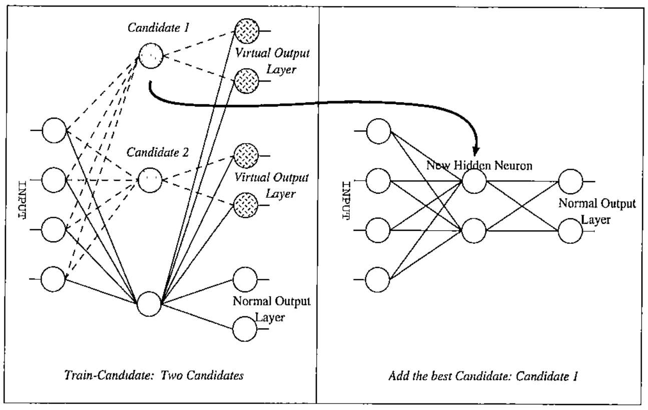


Fig. 2 A pool of two candidate neurons is created temporarily, with the corresponding virtual output layers. The *candidate* 1 is selected and incorporated into the network.

neurons, temporal virtual output layer and all the temporal weights not related to the new hidden neuron are removed from the network. Once the new hidden neuron is inserted in the network in P_{NA}, all the weights are trained to minimize the normal output error in P_{TNN}.

3. Dynamic Constructive Fault Tolerant Algorithm: DCFTA

3.1 Fault Model

Fault tolerance is frequently cited as an important property of NNs, however, the loss of a single weight is frequently sufficient to completely disrupt a learned function. A physically plausible type of fault is the loss of connection between two neurons (*open fault*) [7], this relates to the loss of an arc in a directed graph which abstractly represents the topology of NNs [16]. This fault is equivalent to the case when the synaptic weight is set at 0, which is equivalent to the conventional stuck-at-0 type. This fault is assumed in this paper.

The fault tolerance metric adopted is the percentage of recognized patterns as function of the percentage of faulty weights in the network (which measures how badly the network's performance degrades as a function of the percentage of faulty weights).

The output of the neuron k in the output layer is classified as follows:

$$\begin{aligned} o_k^p &= 1 \quad \text{if} \quad \sigma = \sum_{j=0}^{H} w_{jk} o_j^p > 0, \\ o_k^p &= 0 \quad \text{if} \quad \sigma = \sum_{j=0}^{H} w_{jk} o_j^p < 0. \end{aligned} \tag{8}$$

The output is considered wrong if it switches from 1 to 0 or from 0 to 1, this happens if the sign of σ changes.

3.2 DCFTA

In the previous section, a dynamic constructive learning algorithm DCA is proposed. DCA generates a network with nearly minimum architecture. In this section a fault tolerant algorithm based on DCA will be presented, the algorithm is called dynamic constructive fault tolerant algorithm (DCFTA). Actually the training procedure starts by setting the weights at small random values. During the training the weights are modified/updated to minimize the error function and no fault tolerant mechanism is incorporated. The aim of this research is to build DCFTA which incorporates a fault tolerance mechanism in the training/construction process (P_{TNN}/P_{TC}) in DCA.

Although fault tolerance is frequently cited as an important property of NNs [15], the loss of single weight is frequently sufficient to completely disrupt a learned function. As the networks starts from a set of weights with small values, a loss of single connection (i.e., stuck-at-0 of the corresponding weights) is not likely to influence the network output much in early stage of the learning process (the details are given in the Appendix). To maintain this property, we propose not to update the weights whose relevances are greater than a given threshold, and the weights whose relevances are smaller than the threshold are updated using the backpropagation algorithm.

In this paper the *relevance* $R(w_{ij})$ of a given weight w_{ij} is defined as the maximum error caused at the primary output by the stuck-at-fault of this weight. It is given by

$$R(w_{ij}) = \underset{p\in P,k\in K}{Max} \mid o_k^p(w_{ij}) - o_k^p(w_{ij}^f) \mid, \tag{9}$$

where $o_k^p(w_{ij})$ is the practical output of the kth neuron in the output layer, $o_k^p(w_{ij}^f)$ is the output when the synaptic weight w_{ij} is stuck at a faulty value w_{ij}^f, and $\mid x \mid$ denotes the absolute value of x. The maximum is over the set of all primary output neurons K and the set of all training patterns P. In the same way we define the relevance of a candidate weight w_{ij}^c as follows

$$R(w_{ij}^c) = \underset{p\in P,k\in K}{Max} \mid v_k^p(w_{ij}^c) - v_k^p(w_{ij}^f) \mid, \tag{10}$$

where $v_k^p(w_{ij}^c)$ is the output of the kth neuron of the virtual output layer, $v_k^p(w_{ij}^f)$ is the virtual output when the synaptic weight w_{ij}^c is stuck at a faulty value w_{ij}^f. The relevance $R(w_{ij})$ is an indication of the importance of the weight w_{ij} to the network. Note that the output $v_k^p(w_{ij}^f)$ is compared to the practical output rather than the theoretical output.

The relevance of any weights $R(w_{ij})$ can be evaluated exhaustively by setting w_{ij} to 0 and applying all the training patterns to the NN and evaluating the maximum error at the primary output. However the time for exhaustive evaluation of the relevance can be very long and impractical, since it requires a forward propagation of all the training patterns for each and every synaptic weight.

To avoid the long evaluation time, the relevances are estimated in the training phase using the Taylor expansion of the output around fault-free weights in the same way as in [12], [18].

Now that the effect of the stuck-at-fault is estimated, we propose, in both phases *Train-Normal-Net* (P_{TNN}) and *Train-Candidate* (P_{TC}), to update only the weights which have their relevance less than a specified threshold θ. By doing so, the constructed network will be fault tolerant.

The DCFTA can be summarized as follows:

```
Step 1. (Initialization)
        . initialize the weights of the network with
          single hidden neuron
        . fix the training parameters
        . fix the threshold $\theta$
Step 2. (Train-Normal-Net: $P_{TNN}$)
    for each cycle do
        . calculate the relevance of all weights
        . update all the weights $w_{ij}$ which satisfy
          the condition $R(w_{ij}) < \theta$
        . if the learning converges or the max-
          imum network size is reached, STOP,
          otherwise continue
        . if no improvement after $T_N$ cycle, go to
          Step3, otherwise go to Step 2.
Step 3. (Train-Candidates: $P_{TC}$)
        . create a pool of candidate neurons
        . initialize the weights on the connection
          to and from the candidate neurons
    1: for each cycle do
        . calculate the relevance of the candi-
          date weights $w_{ij}^c$
        . update the weights $w_{ij}^c$ which satisfy
          the condition $R(w_{ij}^c) < \theta$
        . if no improvement after $T_{cand}$ cycle, go to
          Step 4, otherwise go to 1
Step 4. (Neuron Addition: $P_{NA}$)
        . select the best candidate neuron $C_b$ pro-
          ducing the minimum error $E_{cand}$
        . add this neuron to the original network,
          then go to Step 2.
```

4. Experimental Results

In this section the proposed DCFTA is evaluated. It is shown that DCFTA produces a network that exhibits better fault tolerance abilities. All the experiments reported in this paper were run on Sun UltraSparc AS 7000 workstation.

4.1 Mechanical Parts Classification

This problem consists of classification of mechanical parts into seven classes based on similarity feature[17]. The training set consist of 19 mechanical parts, each part is presented on 6×9 pixels. The DCFTA starts with a network with a single hidden neuron, then the hidden neurons are added one by one following the algorithm given in the previous section. The learning stops when the network is able to classify all the 19 parts. The initial weights are randomly set to values that are uniformly distributed in $[-0.5,0.5]$. A pool of five candidates is used during the DCFTA.

As the results depend on the weights set from which the training phases start, the algorithm was run hundred times for each value of the threshold θ. In this experiment, the number of hidden neurons generated by the DCFTA are almost two times larger than that generated by DCA (i.e., it has double number of hidden neurons). For fair fault tolerance comparison, we investigate the networks with the same size (i.e., same number of hidden neurons). In the case of mechanical parts classification problem, the network with 6 and 8 hidden neurons are investigated.

M.D. Emmerson et al. proposed to train a network with small number of hidden neurons then construct an *augmented* network by duplicating the hidden neurons[7]. Since the DCFTA is a constructive method, we propose to compare it to the method of M.D. Emmerson et al. The networks with 3 and 4 hidden neurons are trained, then the *augmented* NNs with 6 and 8 hidden neurons are constructed. After the training has been finished, the network's tolerance to damage is assessed. As the results depend on the weights from which the training process is initiated[13],[14], hundred experiments were made for each network with different initial weights.

The fault tolerance of the networks is assessed by setting at fault a number of randomly selected weights, then the patterns of the training set are applied, and the percentage of recognized patterns is assessed. Since some links are more significant than others, the process was repeated 200 times for each number of faulty weights and the results are averaged.

The simulation results are presented in Fig. 3 and Fig. 4 for the networks 54-6-3 and 54-8-3, respectively, with $\theta = 0.2$ and $\theta = 0.3$, where n_I-n_H-n_O denotes a NN with n_I, n_H, and n_O neurons in the input layer, hidden layer, and output layer, respectively. The *Consθ* represents the NN generated by DCFTA. The *FixAug* represents the NNs 54-3-3 (respectively 54-4-3) trained by BP algorithm, then *augmented*[7]. *ConsAug* represents the NNs 54-3-3 (respectively 54-4-3) constructed by DCA, then *augmented*, and for reference, we plot $\theta = \infty, H_0 = 6$ (respectively $\theta = \infty, H_0 = 8$) which represents the networks with prefixed architecture 54-6-3 (respectively 54-8-3) trained with standard backpropagation algorithm. The results show that the recognition rate of the NNs degrades faster as the value of θ increases. It can be seen that, with $\theta = 0.2$ almost the best fault tolerant networks among NNs with the same number of hidden neurons are obtained. It can be seen also that constructed then augmented networks are less fault tolerant. It is possible to use a smaller value of θ, and get better fault tolerant NN, however, with a smaller value, the build in NN becomes larger, and the comparison with smaller network is not fair.

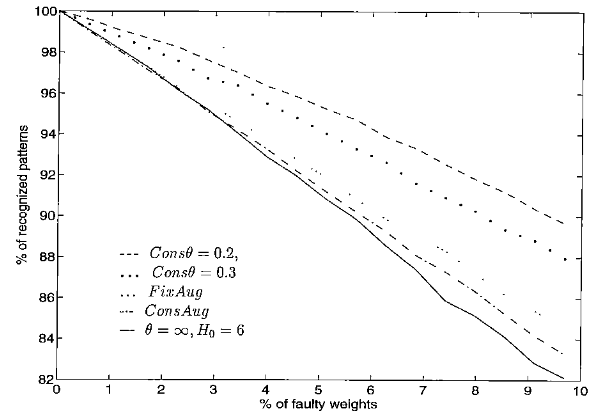


Fig. 3 The percentage of classified parts as function of the % of faulty weights, by 54-6-3 networks.

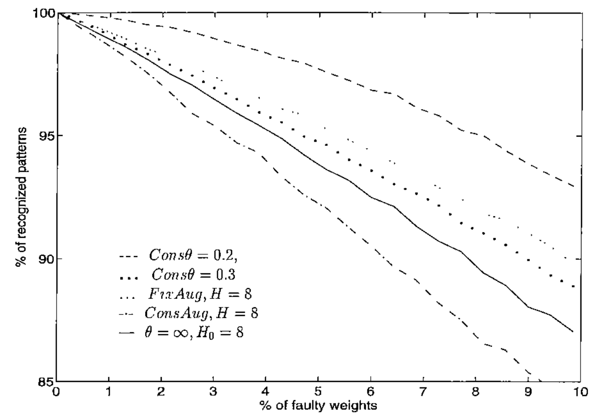


Fig. 4 The percentage of classified parts as function of the % of faulty weights, by 54-8-3 networks.

4.2 Characters Recognition Problem

The second application consists of characters recognition, the 26 characters from A to Z presented on 7×7 binary image plane are considered. The networks have 49 neurons in the input layer, and 26 neurons in the output layer.

The DCFTA algorithm was run hundred times for each value of the threshold θ. As expected, in many times the network generated by the DCFTA is larger when the fault tolerant mechanism is incorporated. The number of hidden neurons built into the network varied

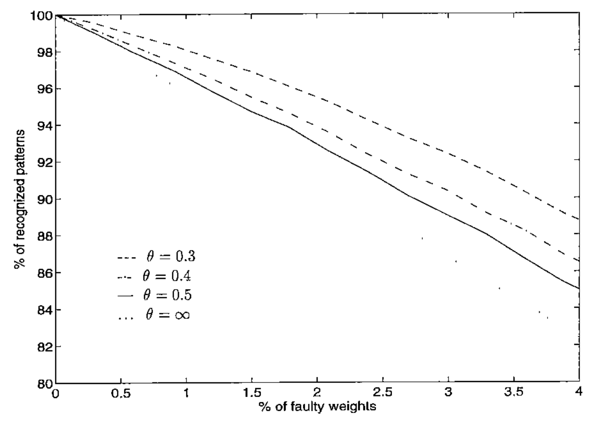


Fig. 5 The percentage of recognized characters as function of the % of faulty weights.

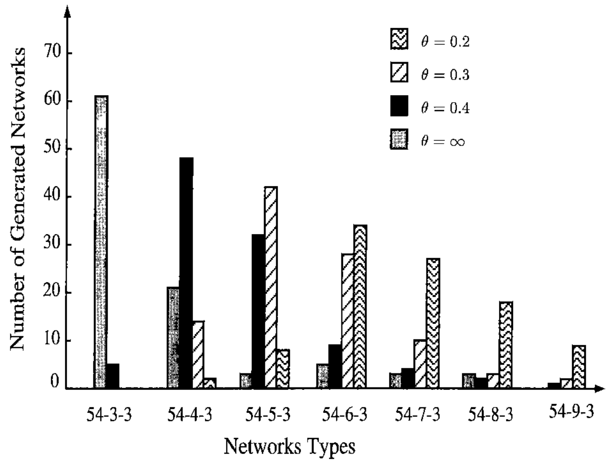


Fig. 6 The number of networks architecture generated (constructed) by DCFTA with different value of θ.

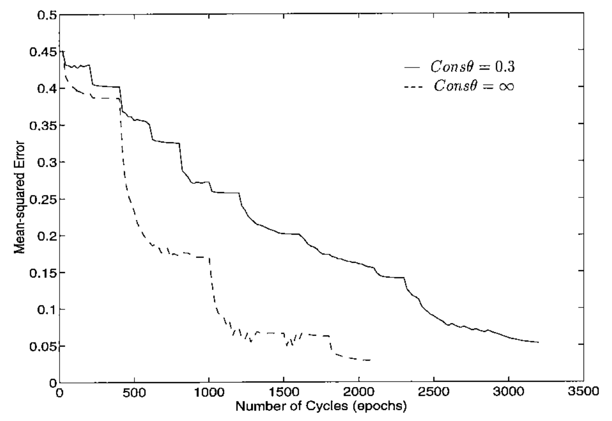


Fig. 7 The mean-squared error as function of the learning cycles for both $\theta = \infty$ and $\theta = 0.3$ in the case of mechanical parts classification problem.

between 8 and 13 with a majority of 9 hidden neurons. To compare objectively the fault tolerance of the networks, we investigate the networks with the same size, that is the network with 9 hidden neurons.

After the training has been finished, the network's tolerance to damage is assessed.

The simulation results are presented in Fig. 5 for the NNs with different values of θ. $\theta = \infty$ represent the NN constructed without any constraint (i.e. no fault tolerant mechanism). The results show that the recognition rate of the NNs degrades faster as the value of θ increases. The best fault tolerant network is obtained when $\theta = 0.3$. It is possible to use a smaller value of θ, however, with a smaller value, the build in NN becomes larger and it is difficult to get a network with 9 hidden neurons.

5. Analysis

In this section, the effect of the proposed fault tolerant mechanism on the size of generated networks and the output error during training is analyzed.

5.1 Network Size

By forcing the weights to have a relevance less than a specified threshold, the DCFTA generates networks with larger number of hidden neurons. It is interesting to compare the size of the generated networks as function of the threshold θ. Figure 6 shows the number of networks generated by the proposed DCFTA with different value of θ in the case of mechanical parts classification problem. For each value of θ, a hundred NNs are generated, and the figure shows how many NN types with a certain number of hidden neurons are generated. It can be seen that, when θ is set to a small value, the size of the networks tend to be large, and if θ has a large value, the generated networks tend to be of small size and less fault tolerant. It is suggested to use a medium threshold $\theta \in [0.2, 0.3]$ to construct a fault tolerant network with reasonable size.

5.2 The Output Error

We assumed that whenever no significant error reduction has occurred after a given number of cycles T_N, there must be a residual error that should be reduced. To achieve this, an independent neuron (a *candidate*) is dynamically created, then inserted into the network. It is interesting to analyze the effect of the insertion of a new hidden neuron on the mean-squared error. Figure 7 presents the mean-squared error as function of the learning cycles for both $\theta = \infty$ (DCA) and $\theta = 0.3$ (DCFTA) in the case of mechanical parts classification problem. It can be seen, specially for $\theta = \infty$, how the output error changes when a new hidden neuron is inserted. From Fig. 7 we can realize also that using the fault tolerant mechanism with $\theta = 0.3$ increases the number of cycles by almost 50%.

6. Conclusion

In this paper, a new dynamic constructive fault tolerant algorithm (DCFTA) for neural network is proposed. The algorithm starts with a network with single hidden neuron, and a new hidden neuron is added dynamically to the network whenever it fails to converge (i.e. learn the task). To generate a fault tolerant network, the relevance of each synaptic weight is estimated in each cycle, and only the weights which have their relevance less than a specified threshold are updated in that cycle. The simulation results indicate that the networks constructed by DCFTA have a significant fault tolerance abilities.

Since it is difficult in practical applications to know a priori the network's size which can learn the task, the proposed constructing algorithm presents a tool to construct dynamically a nearly minimal network, with better fault tolerance property, starting from a net with a single hidden neuron.

References

[1] N.C. Hammadi and H. Ito, "Noise injection into hidden neurons: A learning technique for enhancing the performance of feedforward neural networks," Proc. Int. Conference on Neural Networks and Their Applications (NEURAP'97), pp.245–251, March 1997.

[2] E.B. Bartlett, "Dynamic node architecture learning: An information theoretic approach," Neural Networks, vol.7, no.1, pp.129–140, 1994.

[3] S.E. Fahlman and C. Lebier, "The cascade-correlation learning architecture," Advanced in Neural Information Processing Systems II, pp.524–532. 1989.

[4] S.E. Fahlman and C. Lebier, "Recurrent Cascade-Correlation Architecture," Advanced in Neural Information Processing Systems III, Morgan Kaufman Publishers, pp.190–196. 1991.

[5] E. Littman and H. Ritter, "Cascade network architectures," Proc. Int. Joint Conference On Neural Networks II, pp.398–404, 1992.

[6] E. Littman and H. Ritter, "Cascade LLM Networks," in Artificial Neural Networks II, Elsevier Science Publishers, pp.253–257, 1992.

[7] M.D. Emmerson and R.I. Damper, "Determining and improving the fault tolerance of multilayer perceptions in a pattern-recognition application," IEEE Trans. Neural Networks, vol.4, no.5, pp.788–793, 1993.

[8] J. Nijhuis, B. Hofflinger, A. Schaik, and L. Spaanenburg, "Limits to fault-tolerance of a feedforward neural network with learning," Digest of Fault Tolerant Computing Symposium, pp.228–235, June 1990.

[9] A.F Murray and P.J. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," IEEE Trans. Neural Networks, vol.5, no.5, pp.792–802, 1994.

[10] D.S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," IEEE Trans. Neural Networks, vol.6, no.2, pp.446–456, March 1995.

[11] H. Ito and T. Yagi, "Fault tolerant design using error correcting code for multilayer neural networks," IEEE Int. Workshop on Defect and Fault Tolerance in VLSI systems, pp.177–184, 1994.

[12] N.C. Hammadi and H. Ito, "A learning algorithm for fault tolerant feedforward neural networks," IEICE Trans. Inf. & Syst., vol.E80-D, no.1, pp.21–27, Jan. 1997.

[13] L.F.A. Wessels and E. Barnard , "Avoiding false local minima by proper initialization," IEEE Trans. Neural Networks, vol.3, no.6, pp.899–905, Nov. 1992.

[14] N.C. Hammadi and H. Ito, "On the activation function and fault tolerance in feedforward neural networks," Proc. Int. Workshop on Dependability in Advanced Computing Paradigms, pp.29–34, June 1996.

[15] Y. Tan and T. Nanya, "Fault-tolerant back-propagation model and its generalization ability," Digest IJCNN, pp.2516–2519, 1993.

[16] G. Bolt, "Fault models for artificial neural networks," Digest IJCNN, pp.1373–1378, 1991.

[17] N.S. Merchawi, S.T. Kumara, and C.R. Das, "A probabilistic model for the fault tolerance of multilayer perceptions," IEEE Trans. Neural Networks, vol.7, no.1, pp.201–205, Jan. 1996.

[18] N.C. Hammadi and H. Ito, "A learning algorithm for fault tolerant feedforward neural networks," IEICE Technical Report, FTS95-78, Feb. 1996.

Appendix

In [18], we have derived the relevance as function of the weights, the output of both hidden and output neurons, and the network's input. In the following, it will be shown that when the weights have small values, a loss of single connection, which is equivalent to stuck-at-0 of the corresponding weight, is not likely to influence the network output.

Using the Taylor expansion to the second order as in [12] and [18], the maximum error caused at the primary output neurons, when w_{ij} is stuck-at-0, is the relevance of w_{ij}, and it is given by the following equation.

$$R(w_{ij}) = Max\left| -w_{ij}\frac{\partial o_k^p}{\partial w_{ij}} + \frac{w_{ij}^2}{2}\frac{\partial^2 o_k^p}{\partial w_{ij}^2}\right|, \qquad \text{(A· 1)}$$

where o_k^p is the kth practical output which is a multivariable function depending on the weights. All the differentials in Eq. (A· 1) can be calculated as functions of the outputs of hidden neurons and the network's output, which are all available during training.

In the following x_i^p, y_j^p and o_k^p denote the output of ith neuron in the input layer, the output of jth neuron in the hidden layer and the output of kth neuron in the output layer for the pth pattern, respectively. u_{ij} an *input-to-hidden* weight and w_{jk} a *hidden-to-output* weight.

The sigmoid activation function has the following properties:

$$f'(x) = f(x)(1 - f(x)). \qquad \text{(A· 2)}$$

$$0 < f(x) < 1 \quad \text{for any value of } x. \qquad \text{(A· 3)}$$

Without loss of generality, let's calculate the relevance of a particular *hidden-to-output* weight $w_{j_0k_0}$ and a particular *input-to-hidden* weight $u_{i_0j_0}$.

A.1 In the Case of *Hidden-to-Output* Weight $w_{j_0k_0}$

The relevance of $w_{j_0k_0}$ can be written as follows [18]:

$$R(w_{j_0k_0}) = Max\left| -w_{j_0k_0} y_{j_0}^p o_{k_0}^p (1 - o_{k_0}^p) - \frac{w_{i_0j_0}^2}{2} y_{j_0}^{p2} o_{k_0}^p (1 - o_{k_0}^p)(1 - 2o_{k_0}^p) \right|. \quad \text{(A· 4)}$$

Then we can write the following inequality

$$R(w_{j_0k_0}) < | w_{j_0k_0} | Max | y_{j_0} o_{k_0}^p (1 - o_{k_0}^p) | + \frac{w_{i_0j_0}^2}{2} Max | y_{j_0}^{p2} o_{k_0}^p (1 - o_{k_0}^p)(1 - 2o_{k_0}^p) | . \quad \text{(A· 5)}$$

All the neuron's outputs are generated using the sigmoid function, thus the satisfy the following condition $0 < y_j < 1$ and $0 < o_k^p < 1$. Using this property the above inequality can be written as follows

$$R(w_{j_0k_0}) < | w_{j_0k_0} | Max | o_{k_0}^p (1 - o_{k_0}^p) | + \frac{w_{i_0j_0}^2}{2} Max | o_{k_0}^p (1 - o_{k_0}^p)(1 - 2o_{k_0}^p) | . \quad \text{(A· 6)}$$

With simple calculation we find that $Max \mid o_{k_0}^p (1 - o_{k_0}^p) \mid = 0.25$ and $Max \mid o_{k_0}^p (1 - o_{k_0}^p)(1 - 2o_{k_0}^p) \mid \cong 0.0963$. Then the relevance of $w_{j_0k_0}$ satisfies the following inequality

$$R(w_{j_0k_0}) < 0.25 \mid w_{j_0k_0} \mid +0.0963 \frac{w_{i_0j_0}^2}{2}. \quad \text{(A· 7)}$$

It is clear that if the weigh $w_{j_0k_0}$ has a small value, the relevance is small.

A.2 In the Case of *Input-to-Hidden* Weight $u_{i_0j_0}$

The relevance can be written as follows:

$$R(u_{i_0j_0}) = Max| -u_{i_0j_0} x_{i_0}^p w_{j_0k} y_{j_0}^p (1 - y_{j_0}^p) o_k^p (1 - o_k^p) - \frac{u_{i_0j_0}^2}{2} x_{i_0}^{p2} w_{j_0k} y_{j_0}^p (1 - y_{j_0}^p)(1 - 2y_{j_0}^p) o_k^p (1 - o_k^p) - \frac{u_{i_0j_0}^2}{2} x_{i_0}^{p2} w_{j_0k}^2 y_{j_0}^{p2} (1 - y_{j_0}^p)^2 o_k^p (1 - o_k^p)(1 - 2o_k^p) | . \quad \text{(A· 8)}$$

In the same way as for $w_{i_0j_0}$, using the above equation, we can write the following inequality

$$R(u_{i_0j_0}) < 0.0625 \mid -u_{i_0j_0} \mid Max \mid x_{i_0} w_{j_0k} \mid +0.0241 \frac{u_{i_0j_0}^2}{2} Max \mid x_{i_0}^{p2} w_{j_0k} \mid +0.00602 \frac{u_{i_0j_0}^2}{2} Max \mid x_{i_0}^{p2} w_{j_0k}^2 \mid . \quad \text{(A· 9)}$$

It can be realized from the Eqs. (A·8) and (A·9), that when the weights have a small values, the stuck-at-0 of a single weight relevance $w_{j_0k_0}$ or $u_{i_0j_0}$ is likely to produce a small relevance, that is a small deviation of the network output.

The same calculation can be made for the relevance of candidate weights.

A.3 In the Case of *Candidate-to-Output* Weight w_{ck_0}

In the same way as in A.1, the following equation can be written

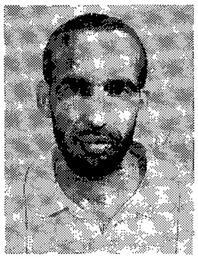
$$R(w_{ck_0}) < 0.25 \mid w_{ck_0} \mid +0.0963 \frac{w_{cj_0}^2}{2}. \quad \text{(A· 10)}$$

A.4 In the Case of *Input-to-Candidate* Weight u_{i_0c}

In the same way as in A.2, the following equation can be written

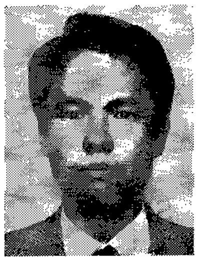
$$R(u_{i_0c}) < 0.0625 \mid -u_{i_0c} \mid Max \mid x_{i_0}^p w_{ck} \mid +0.0241 \mid \frac{u_{i_0c}^2}{2} Max \mid x_{i_0}^{p2} w_{ck} \mid +0.00602 \frac{u_{i_0c}^2}{2} Max \mid x_{i_0}^{p2} w_{ck}^2 \mid . \quad \text{(A· 11)}$$

It can be realized from the Eqs. (A·10) and (A·11), that when the candidate weights have a small values, the stuck-at-0 of a single weight relevance w_{ck_0} or u_{i_0c} is likely to produce a small relevance, that is a small deviation of the network output.



Nait Charif Hammadi was born in Tinghir, Morocco, on December 25, 1965. He received the Engineer diploma in electronics and control, from Ecole Hassania des Traveaux Publics, Casablanca, Morocco, in 1990. In 1991, he joined the Ecole Superieure de Technologie, Mohamed I University, as lecturer. Currently, he is working toward his Ph.D. in Information and Computer Sciences, Graduate School of Science and Technology, Chiba university, Japan. His research interests include neural networks and fault tolerant systems. He is member of IEEE.

Toshiaki Ohmameuda was born in Tochigi, Japan in 1967. He received the B.E., M.E. and Dr. of Engineering in electronic engineering from the University of Tokyo in 1989, 1991 and 1994, respectively. He joined the faculty of Chiba University as Research Assistant since 1994. His research interests are VLSI testing, VLSI design and fault tolerant systems. Dr. Ohmameuda is a member of the Japan Society of Applied Physics.



Keiichi Kaneko was born in Tokyo, Japan, on November 13, 1962. He received the B.E., M.E, Ph.D. degrees from the University of Tokyo in 1985, 1987, 1994, respectively. In 1987, he joined the University of Tokyo as a research associate. Now, he is a lecturer of Chiba University. His main research areas are software engineering, parallel and distributed computation, and fault tolerant systems. He is a member of ACM, IPSJ, and JSSST.

Hideo Ito was born in Chiba, Japan, on June 1, 1946. He received the B.E. degree from Chiba University in 1969, and the D.E. degree from Tokyo Institute of Technology in 1984. He joined Nippon Electric Co. Ltd. in 1969, and Kisarazu Technical College in 1971. Since 1973 he has been a member of the Chiba University. He is currently a Professor of Department of Information and Computer Sciences. His research interests include easily testable design, test generation, VLSI architecture, fault-tolerant design of parallel & distributed systems, and defect-tolerant VLSI design. He is a member of the Information Processing Society of Japan and the IEEE Computer Society.