Relationship between fault tolerance, generalization and the Vapnik–Chervonenkis (VC) dimension of feedforward ANNs.

Dhananjay S. Phatak
Electrical Engineering Department
State University of New York, Binghamton, NY 13902-6000

(*Proceedings of IJCNN, Washington DC, Aug. 1999*)

Abstract

It is demonstrated that Fault tolerance, generalization and the Vapnik–Chervonenkis (VC) dimension (which is in turn related to the intrinsic capacity/complexity of the ANN) are inter-related attributes. It is well known that the generalization error if plotted as a function of the VC dimension h, exhibits a well defined minimum corresponding to an optimal value of h, say h_{opt}.

We show that if the VC dimension h *of an ANN satisfies* $h \leq h_{\text{opt}}$ *(i.e., there is no excess capacity or redundancy), then Fault Tolerance and Generalization are mutually conflicting attributes. On the other hand, if* $h > h_{\text{opt}}$ *(i.e., there is excess capacity or redundancy, then fault tolerance and generalization are mutually synergistic attributes.*

In other words, training methods geared toward improving the fault tolerance can also lead to better generalization and vice versa, only when there is excess capacity or redundancy. This is consistent with our previous results indicating that complete fault tolerance in ANNs requires a significant amount of redundancy.

I. Introduction

Despite their gross similarity with biological systems, ANNs are not necessarily fault tolerant, as indicated by results in the literature [1, 2]. Contrary to the expectation, fault tolerance in ANNs comes at the cost of a significant amount of redundancy [2]. Furthermore, merely providing redundancy is not enough; the learning algorithms also must be modified to utilize the available redundancy in order to achieve fault tolerance [2]. Several researchers have experimented with modified training algorithms that enhance fault tolerance of feedforward networks (for example [3, 4], etc).

Relation between fault tolerance and generalization has also been looked at [5, 6]. Most results reported in the literature indicate that incorporating fault tolerance enhancement during training also leads to better generalization and vice versa. This needs to be interpreted carefully because redundancy might be detrimental to generalization performance. It is known that the "simplest hypothesis/model is the least likely to overfit". For example, the net that uses the least number of independent parameters (which include the weights and biases) to achieve a given mapping is least likely to overfit the data and is therefore likely to have good generalization performance. If one adds redundancy (in the form of extra units or additional independent parameters) in order to achieve fault tolerance, it is likely to hurt generalization performance because more than the necessary number of parameters is being utilized to achieve the same mapping. In this sense, addition of redundancy to achieve fault tolerance would appear to trade off generalization performance.

If, however, the number of parameters is fixed and one imposes additional constraints to encourage fault tolerance, the resulting net might generalize better than a net generated without the fault tolerance constraints. For instance, if the fault tolerance is enhanced by adding extra terms to the objective function, in some sense, it adds a "spread" to the mapping being learned: the error between the extra terms and the outputs is also being minimized along with the original mapping. Thus, the parameters available can be thought to be "stretched" as far as possible to fit the original mapping as well as the extra terms. Such a spread could conceivably enhance the generalization performance because the net could handle some unseen input if its effect is equivalent to a fault that the net has learned to tolerate. Injecting faults during training achieves a similar effect.

In this paper we explore the inter-relationships between fault tolerance, generalization and intrinsic capacity/redundancy. The next section presents qualitative semi-rigorous arguments based on learning theory to demonstrate that when there is excess capacity, fault tolerance and generalization are mutually synergistic. On the other hand,

if there is no excess capacity or redundancy then fault tolerance and generalization are mutually conflicting requirements. Section III shows simulation data which corroborates the analytical findings.

II Fault tolerance, generalization and the Vapnik–Chervonenkis (VC) dimension

Let the target function to be learned by the ANN be $f(\bar{x})$ ((vectors are denoted by an overbar such as $\bar{x}$). For the sake of simplicity a single-output considered without loss of generality (i.e., the same analysis applies to multi-output mappings as well). Note that $f(\bar{x})$ includes I/O mappings for both approximation as well as classification tasks. Assume that the network realizes an approximation $F(\bar{x}, \bar{w})$ instead of the target function $f(\bar{x})$. In the domain of learning theory, the expected value of discrepancy or loss is defined by the *risk functional* [7]

$$R(\bar{w}) \ = \int \mathcal{D}(f(\bar{x}), F(\bar{x}, \bar{w}))\ dP(x, f(\bar{x})) \tag{1}$$

where $dP(x, f(\bar{x}))$ is the joint probability distribution of input vector $\bar{x}$ and the desired response $f(\bar{x})$; $\mathcal{D}$ is a distance (such as absolute value or L_2 norm. in general, it can be a functional mapping a function into a real number); and the integral is taken in the *Riemann–Stieltjes sense* [8]. The goal of the learning process is to minimize the risk functional $R(\bar{w})$ over the class of functions $F(\bar{x}, \bar{w})$. However, the joint probability distribution $P(x, f(\bar{x}))$ is unknown and hence $R(\bar{w})$ cannot be evaluated directly. To overcome this difficulty, the method of risk minimization constructs the *empirical risk functional*

$$R_{\text{emp}}(\bar{w}) \ = \frac{1}{N} \sum_{i=1}^{N} \mathcal{D}\ (f(\bar{x}_i), F(\bar{x}_i, \bar{w})) \tag{2}$$

where $\bar{x}_i$ are iid training samples. The principle of empirical risk minimization [9, 7] states that the weight vector $\bar{w}^*$ which minimizes the empirical risk $R_{\text{emp}}(\bar{w})$ guarantees that $R(\bar{w}^*)$ converges in probability to the minimum possible value of actual risk $R(\bar{w})$ as $N \rightarrow \infty$, provided the functional $R_{\text{emp}}(\bar{w})$ converges uniformly to the actual risk functional $R(\bar{w})$ [8].

The risk functional $R(\bar{w})$ can be thought to approximate the *average generalization error*, denoted by $E_G(\bar{w})$ (the approximation becomes exact for classification tasks for which the outputs can be deemed to be binary [8]), while the empirical risk functional $R_{\text{emp}}(\bar{w})$ represents the *average training error*, denoted by $\nu_{\text{train}}(\bar{w})$. For a set of classification functions with VC dimension h, the following inequality holds [9, 7]:

$$\text{Prob}\{\sup_{\bar{w}} |E_G(\bar{w}) - \nu(\bar{w})| > \epsilon\} \quad < \quad \alpha = (\frac{2eN}{h})^h exp(-\epsilon^2 N) \tag{3}$$

where e is the base of the natural logarithm. From the above equation, it is possible to derive the general bound [8]

$$E_G(\bar{w}) < \nu_{\text{max}}(\bar{w}) \ = \ \nu_{\text{train}}(\bar{w}) \ + \rho(N, h, \alpha, \nu) \tag{4}$$

where ρ is a confidence interval Thus the average generalization error is guaranteed to be less than the sum of two competing terms in the right hand side of equation (4) above [7]. For a fixed number of samples N, the first term $\nu_{\text{train}}(\bar{w})$ decreases monotonically as the capacity or the VC dimension h is increased, while the confidence interval $\rho(N, h, \alpha, \nu)$ increases monotonically with h. Hence, their sum $\nu_{\text{max}}(\bar{w})$ is a concave function of h and goes through a minimum corresponding to an optimum value h_{opt} of the VC dimension h as illustrated in Figure 1.

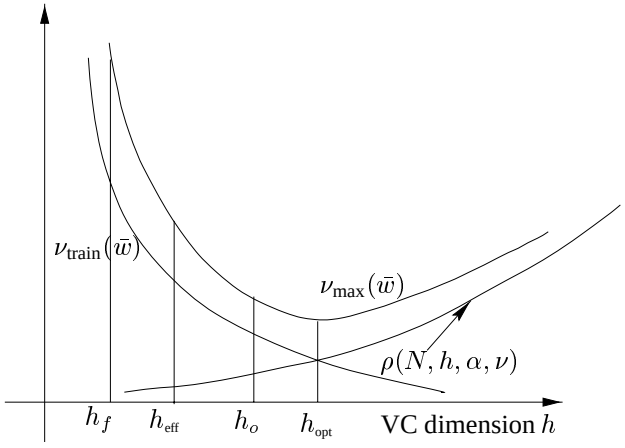

Figure 1 : Illustration of the dependence of $\nu_{\text{train}}(\bar{w})$, $\rho(N, h, \alpha, \nu)$ and $\nu_{\text{max}}(\bar{w})$ on the VC dimension h.

To understand the relation with fault tolerance, assume that the original non-faulty network has a VC dimension h_o as shown in Figure 1. Since most faults can be modeled as stuck-at faults [2] affecting the parameters of the network (weights, biases or unit outputs), a "faulty" network can be thought to have fewer independently adjustable parameters available (since the faults cause some parameters get "stuck-at" fixed values, making them unavailable for adaptation). Hence, the capacity or **VC dimension h_f of the "faulty net" satisfies the inequality:** $h_f \leq h_o$ (5)

as illustrated in Figure 1.

The effect of including extra terms in the objective function to enhance the fault tolerance or regularize complexity, is in some sense, also equivalent to training a net with a VC dimension h_{eff} that satisfies

$$h_{\text{eff}} \leq h_o \tag{6}$$

as shown in Figure 1. The reason is the following: when the extra terms are included in the objective function, the available parameters are utilized to minimize both the the original error terms and the extra terms. Consequently, the values the parameters can assume are more restricted. In fact the main purpose behind the extra terms is to "guide the search" to specific solutions which in turn implies that these terms "restrict" the values the independent parameters can assume or equivalently, reduce the degrees of freedom. Such a reduction in the degrees of freedom is in turn equivalent to a reduction in the capacity or the VC dimension.

However, the reduction in the VC dimension as a result of adding constraints (via direct constrained optimization or modifying the objective function by adding extra terms) is unlikely to be as big as the reduction in capacity due to stuck-at faults which simply make some parameters unavailable. Hence h_{eff} must be in-between h_f and h_o as shown in Figure 1.

$$h_f \leq h_{\text{eff}} \leq h_o \tag{7}$$

Having established relation (7) above, we can infer the following from Figure 1:

(a) If $h_o \leq h_{\text{opt}}$, i.e., the number of parameters is less than that required for best generalization performance, then incorporating constraints to enhance fault tolerance or regularize complexity (by direct constrained optimization or by adding the extra terms to the objective function or by any other method) leads to an effective VC dimension h_{eff} which is still smaller. Hence, the generalization error increases.

(b) On the other hand, if $h_o > h_{\text{opt}}$, then constraining parameters to enhance fault tolerance or regularize complexity will lead to an h_{eff} that is closer to the optimal value h_{opt}, which will lead to a *decrease* in the generalization error. In summary

If $(h_o \leq h_{\text{opt}})$ then /* no overcapacity/redundancy */
    fault tolerance and generalization are mutually
    conflicting attributes.
else for $(h_o > h_{\text{opt}})$ /* overcapacity exists */
    fault tolerance, complexity regularization and
    generalization are synergistic.
endif

The above results conform to intuition: if the number of parameters available is small, then all of them should be dedicated to learning the training set. Adding fault tolerance criteria in this situation will only worsen generalization performance. On the other hand, if there are excess parameters, then adding fault tolerance criteria "constrains" their use, thereby improving generalization.

Injecting faults during training is also equivalent to reducing the effective VC dimension, because the stuck-at faults make some parameters unavailable, thereby reducing the degrees of freedom. If $h_o > h_{\text{opt}}$ then injecting faults during training can enhance both fault tolerance and generalization, which would explain the results reported by many researchers. These findings corroborate our previous results indicating that complete fault tolerance in ANNs necessarily implies redundancy.

III Simulation Results and Discussion

The above result can be verified experimentally (through simulations). For the purpose of illustration we report results for a 2 class distinction problem because it is easy to visualize the task to be learned and the minimum capacity/size (number of nodes, connections, etc.) required to successfully learn the task is also known. The task simply consists of distinguishing between inputs belonging to two different classes as shown in Figure 2 on the next page. As seen in figure 2, the data consists of points that are uniformly distributed within the unit square which is divided into two classes by two lines L_1 and L_2. The "area" of the unit square is split equally between the two classes. Training data set consists of 50 points, out of which 12 points (including the 8 points illustrated by solid circles and squares) were hand-picked to "delineate" the position of decision boundaries. Remaining points in the training data set were randomly generated to be uniformly distributed within the unit square. Test data set consisted of 500 additional uniformly distributed points within the unit square.

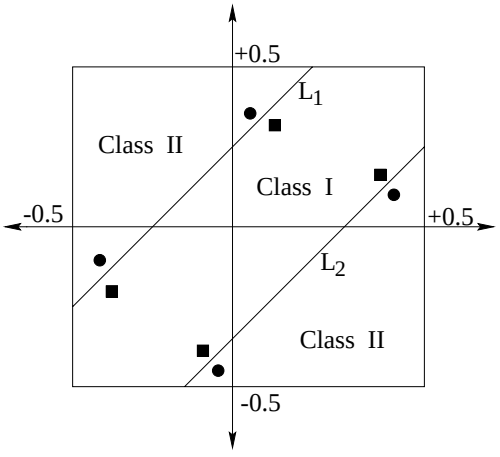

Figure 2: Two class distinction task which is used to experimentally illustrate the inter relationships between fault tolerance, capacity/redundancy and generalization.

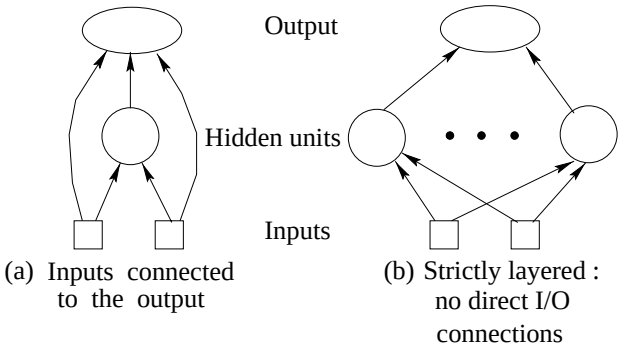

Figure 3: Network architectures for the 2 class distinction task (2 inputs and 1 output).

Network architectures for this task are illustrated in Figure 3. If direct input/output connections are allowed as in Figure 3-(a), then just one hidden unit suffices to learn this task. If such direct I/O connections are not allowed making the network strictly layered, then at least two hidden units are required to learn this task. These are the "minimum" sized network(s) required to learn this task. Knowing these minimum required sizes (i.e., minimum required capacity) makes it is possible to select networks with less than the necessary size, networks with the required minimum size and networks having lager than minimum sizes; in order to corroborate the above analysis.

For simulations, five network architectures with different sizes covering the entire range (under, par and over the required size/capacity) were selected. Each architecture was trained by two algorithms

(i) one included extra terms in the objective function being minimized in order to enhance the fault tolerance. These terms capture the effect of injecting faults during training. (Outputs of hidden units are set stuck-at $\pm$Max. The resulting error represents deviation from nominal outputs due to faults. This error is also included in the objective function along with the normal training error in order to enhance fault tolerance.)
(ii) normal error minimization training without fault tolerance enhancement terms in the objective function.

For each architecture and each training method, 100 trials were run (with random initial weights for each trial) and the average generalization error (for the test data set) over the 100 runs was calculated. The results are summarized in Table 1 (on the next page).

For some architectures, in some of the runs, the net could not learn the training data set successfully. In such cases, average generalization error is also calculated over the successful runs only. The table shows that when there is excess capacity to begin with (cases with 10 and 4 hidden units), incorporating fault tolerance criteria during training improves generalization as well. When the capacity is the minimum required, however, adding fault tolerance terms to the objective function causes the algorithm to get stuck at local minima, leading to unsuccessful runs. This leads to a substantially higher generalization error. Note that even when the generalization error over only successful runs is compared, the improvement (in generalization error as result of the fault tolerance constraints) is much smaller. In fact the improvement in these cases is (arguably) insignificant. Finally when there is only one hidden unit without the direct I/O connections, the net cannot learn the task at hand. The table indicates that adding fault tolerance constraints during training in this case clearly worsens the generalization error.

Finally, note that the results presented are conservative or "worst case" in some sense. This is because 12 points in the training data set are "hand crafted" to clearly delineate the class boundaries. Hence, if a network can learn this data set at all, it is highly likely to show good generalization as well. In reality, available training data sets are not likely to demarcate the class boundaries so precisely. Furthermore, the class boundaries themselves are likely to have a much more complex shape. In such cases, "learning" the training data set will not necessarily lead to a good generalization performance since the data set itself is not so precise and the decision boundaries are complex. Simulations with such data sets will therefore support our analytical findings even more clearly and strongly.

Network Architecture	Training Type					
	Normal Training			Fault Tolerance Enhancement Incorporated in Training		
	Average Gen. Error on test data	Number of successful runs	Average Gen. Error over successful runs	Average Gen. Error on test data	Number of successful runs	Average Gen. Error over successful runs
10 hidden units strictly layered **(over–capacity)**	13.8935	100	13.8935	9.6411	100	9.6411
4 hidden units strictly layered **(over–capacity)**	13.4023	100	13.4023	12.7029	100	12.7029
2 hidden units strictly layered **(min. required capacity)**	13.1422	100	13.1422	19.8318	85	11.9808
1 hidden unit, direct I/O connections **(min. required capacity)**	50.5555	16	13.7757	90.9339	14	13.7604
1 hidden unit strictly layered (not enough to learn the task: **(under–capacity)**	79.2477	0	79.2477	84.7549	0	84.7549

Table 1 : The effect of incorporating fault tolerance enhancement criteria/constraints during training; on the generalization error for network architectures with different capacities. A "successful run" is one where the net correctly learned to classify each point in the training data set to within the specified tolerance.

References

[1] D. S. Phatak and I. Koren, "Fault Tolerance of Feedforward Neural Nets for Classification Tasks," in *Proceedings of International Joint Conference on Neural Nets (IJCNN), Baltimore, MD*, vol. II, pp. 386–391, Jun. 1992.

[2] D. S. Phatak and I. Koren, "Complete and Partial Fault Tolerance of Feedforward Neural Nets," *IEEE Transactions on Neural Networks*, vol. 6, pp. 446–456, Mar. 1995.

[3] C. Chiu, K. Mehrotra, C. Mohan, and S. Ranka, "Modifying training algorithms for improved fault tolerance," in *Proceedings of the IEEE International Conference on Neural Nets (ICNN), Orlando, FL*, vol. 1, pp. 333–338, Jun. 1994.

[4] A. F. Murray and P. J. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Transactions on Neural Networks*, vol. 5, pp. 792–802, Sept. 1994.

[5] Y. Tan and T. Nanya, "Fault tolerant back propagation model and its generalization ability," in *Proceedings of International Joint Conference on Neural Nets (IJCNN), Nagoya, Japan*, vol. 3, pp. 2516–2519, Oct. 1993.

[6] H. Kim and L. Fu, "Generalization and fault tolerance in rule-based neural networks," in *Proceedings of the IEEE International Conference on Neural Nets (ICNN), Orlando, FL*, vol. 3, pp. 1550–1555, Jun. 1994.

[7] V. N. Vapnik, "Principles of Risk Minimization for Learning Theory," in *Advances in Neural Information Processing Systems 4* (J. E. Moody. et.al., editors), pp. 831–838, Morgan Kaufman Publishers, 1992.

[8] S. Haykin, *Neural Networks, A Comprehensive Foundation*. Macmillan College Publishing, 1994.

[9] V. N. Vapnik, *Estimation of Dependences Based on Empirical Data*. Springer–Verlag, Berlin, 1982.