3-1993

A Fault-Tolerant Optimal Interpolative Net

Daniel J. Simon
Cleveland State University, d.j.simon@csuohio.edu

Hossny El-Sherief
TRW System Integration Group

Original Citation

D. Simon and H. El-Sherief. (1993). A Fault-Tolerant Optimal Interpolative Net. IEEE Conference on Neural Networks, 825-830, doi: 10.1109/ICNN.1993.298665.

Fault-Tolerant Training for Optimal Interpolative Nets

Dan Simon and Hossny El-Sherief

Abstract—**The optimal interpolative (OI) classification network is extended to include fault tolerance and make the network more robust to the loss of a neuron. The OI net has the characteristic that the training data are fit with no more neurons than necessary. Fault tolerance further reduces the number of neurons generated during the learning procedure while maintaining the generalization capabilities of the network. The learning algorithm for the fault-tolerant OI net is presented in a recursive format, allowing for relatively short training times. A simulated fault-tolerant OI net is tested on a navigation satellite selection problem.**

I. Introduction

ONE of the difficulties that a neural net trainer often faces is deciding how many neurons to use in the network. If too many neurons are used, training time may be much longer than necessary, and the resultant network may have poor generalization properties [1]. If too few neurons are used, the learning algorithm may not converge to a suitable configuration. It is clearly desirable to use a training method which intelligently and automatically generates the optimal number of neurons.

One solution to this difficulty is the optimal interpolative (OI) net [2]. The OI net is a three-layer classification network which grows only as many middle layer neurons as necessary to correctly classify the training set. The efficient recursive learning procedure presented in [3] and [4] makes the OI net an attractive architecture.

In the present paper we extend the OI net learning algorithm to include fault tolerance. Biological systems are inherently fault tolerant due to the distributed nature of information representation [5]. Fault tolerance has also been touted as an inherent property of artificial neural systems. But this has often been taken for granted rather than being explicitly provided for in the learning method. In this paper we explicitly account for fault tolerance in the choice of the optimal weights. This increases learning time but makes the resulting network more robust to failures.

Section II reviews the architecture of the OI net and the concept of fault tolerance. Section III presents a recursive learning algorithm for a fault-tolerant OI net. Section IV presents some simulation results, and Section V presents concluding remarks.

II. Preliminaries

A. The Optimal Interpolative Net

Suppose we are given a training set with q sets of input–output pairs. Each of the q training inputs $x^i \in R^n$ maps into one of m classes C_j. Let $y^i \in R^m$ be the desired output corresponding to x^i. The output y^i is defined as

$$x^i \in C_j \Longrightarrow y^i = \delta_j \tag{1}$$

where δ_j is the m-dimensional vector containing all zeros except for the jth element, which is one.

The OI net consists of three layers of neurons. The first layer has n neurons, one for each component of the input. The second layer has p neurons, where p is a number which is chosen during training. The third layer has m neurons, one for each component of the output. The weight from the ith input neuron to the jth middle layer neuron is given by V_{ij}, where

$$V = [v^1 \ \cdots \ v^p] \in R^{n \times p}. \tag{2}$$

The vectors v^i are called prototypes and are chosen from the training set inputs during the learning procedure. The activation function at each middle layer neuron is given by $\phi(s) = \exp(s/\rho)$ where ρ is a learning constant chosen by the user. The weight from the jth middle layer neuron to the kth output layer neuron is given by W_{jk}, where W is the weight matrix to be chosen during training

$$\min_W ||Y - W^T G||_F \Longrightarrow W = (GG^T)^{-1} GY^T \in R^{p \times m} \tag{3}$$

where $||\cdot||_F$ refers to the Frobenius norm of a matrix [10]. Hereafter the subscript F will be omitted for convenience. $Y \in R^{m \times q}$ and $G \in R^{p \times q}$ are given by

$$Y = [y^1 \ \cdots \ y^q] \tag{4}$$

$$G = \begin{bmatrix} \phi(v^1, x^1) & \cdots & \phi(v^1, x^q) \\ \vdots & & \vdots \\ \phi(v^p, x^1) & \cdots & \phi(v^p, x^q) \end{bmatrix} \tag{5}$$

where $(\cdot, \cdot)$ denotes the dot product of two vectors. A training input is included as a prototype only if it does not induce ill conditioning in GG^T. This reduces the number of prototypes, and hence limits the number of middle layer neurons in the network.

In practice, the learning procedure is presented with q exemplars during training, one at a time. A given exemplar is included in the minimization problem of (3)–(5) only if it cannot be correctly classified by the network which has been trained up to that point. Those exemplars which are included in Y and G are referred to as subprototypes and are collected in the vectors $\mathbf{z}^i$. So Y and G in (4)–(5) are replaced with

$$Y = [y^1 \ \cdots \ y^l] \tag{6}$$

$$G = \begin{bmatrix} \phi(v^1, z^1) & \cdots & \phi(v^1, z^l) \\ \vdots & & \vdots \\ \phi(v^p, z^1) & \cdots & \phi(v^p, z^l) \end{bmatrix} \tag{7}$$

where l is the number of subprototypes chosen from the exemplar inputs ($l \leq q$).

B. Fault Tolerance

Fault tolerance is a measure of the ability of a system to maintain its functionality in the presence of damage. For a neural network, fault tolerance can be defined as the ability of the network to correctly classify inputs in the presence of a failed neuron.

The OI net is trained to minimize $||Y - W^T G||$ with respect to the weight vector W. A failure of the jth middle layer neuron is equivalent to replacing the jth row of G with zeros. If we assume that all p middle layer neurons are equally susceptible to failure, then we can add fault tolerance to the learning procedure by solving

$$\min_W \left\{ ||Y - W^T G|| + a \sum_{j=1}^{p} ||Y - W^T G_j|| \right\} \tag{8}$$

where G_j is equal to G except that the jth row is replaced with zeros, and a is the relative weight placed on fault tolerance. This problem is in turn equivalent to solving

$$\min_W ||\mathcal{Y} - W^T \mathcal{G}|| \Longrightarrow W = (\mathcal{G}\mathcal{G}^T)^{-1}\mathcal{G}\mathcal{Y}^T \tag{9}$$

where $\mathcal{Y} \in R^{m \times l(p+1)}$ and $\mathcal{G} \in R^{p \times l(p+1)}$ are given by

$$\mathcal{Y} = [Y \quad aY \quad \cdots \quad aY] \tag{10}$$

$$\mathcal{G} = [G \quad aG_1 \quad \cdots \quad aG_p]. \tag{11}$$

III. The Recursive Learning Algorithm

In this section we extend the recursive OI net learning algorithm [3], [4] to include the fault tolerance described in the previous section. We have q exemplars, l subprototypes, and p prototypes such that $q \geq l \geq p$. Denote the initial set of training exemplars by A. We try to classify the exemplar x^i under consideration with the neural network which has been generated so far. If the learning procedure has so far generated p prototypes and l subprototypes, the network mapping is denoted by $f_p^l: R^n \rightarrow R^m$. If x^i can be correctly classified {i.e., $\max[f_p^l(x^i)] = \max(y^i)$} we retain x^i in A and proceed with the next exemplar. If x^i cannot be correctly classified, we remove x^i from A, append it to the matrix Z of subprototypes, and solve the minimization problem (9). We then consider also including x^i as a prototype and appending it to the weight matrix V. For x^i to qualify as a prototype, it must not induce ill conditioning in the matrix $\mathcal{G}\mathcal{G}^T$. This process is repeated until all of the exemplars remaining in A are correctly classified. The notation used in the learning algorithm is summarized in Table I.

TABLE I
OI Net Notation

Symbol	Meaning
x^i	training input vector
n	dimension of each input vector
A	set of all training input vectors
y^i	training output vector
m	dimension of each output vector
q	number of training exemplars
v^i	prototype vector (taken from A)
V	matrix containing prototypes
p	number of prototypes
z^i	subprototype vector (taken from V)
Z	matrix containing subprototypes
l	number of subprototypes
a	fault tolerance weight
f_p^l	the OI neural map based on the p prototypes in V and the l subprototypes in Z

1) Initialization.

$$V = Z = [x^1] \quad v^1 = z^1 = x^1 \quad p = l = 1$$

$$Y_l = [y^1] \qquad \mathcal{Y}_p^l = [Y_l \quad aY_l] \quad n = q - 1.$$

Reindex the exemplars $x^2, \cdots, x^q$ and their corresponding outputs from one to n, place them in the set A, and compute

$$(R_p^l)^{-1} = \frac{1}{\phi^2(v^1, v^1)} \qquad W_p^l = \frac{Y_l^T}{\phi(v^1, v^1)}$$

$$G_p^l = [\phi(v^1, v^1)] \qquad \mathcal{G}_p^l = [G_p^l \quad 0].$$

2) Main Recursion.

For $i = 1$ to n, do the following.

a) Compute $\hat{y}^i = f_p^l(x^i)$. If $x^i \in C_j$ and $\max(\hat{y}_1^i, \cdots, \hat{y}_m^i) = \hat{y}_j^i$, then x^i has been correctly classified by f_p^l. We therefore retain x^i in A and proceed with the next exemplar in A.

b) If f_p^l does not correctly classify x^i, however, we set $z^{l+1} = x^i$ and form the matrices

$$Y_{l+1} = [Y_l \quad y^i] \in R^{m \times (l+1)} \tag{12}$$

$$\mathcal{Y}_p^{l+1} = [Y_{l+1} \quad aY_{l+1} \quad \cdots \quad aY_{l+1}] \in R^{m \times (l+1)(p+1)} \tag{13}$$

$$k_p^{l+1} = [\phi(v^1, z^{l+1}) \cdots \phi(v^p, z^{l+1})] \in R^{p \times 1} \tag{14}$$

$$G_p^{l+1} = [G_p^l \quad k_p^{l+1}] \in R^{p \times (l+1)} \tag{15}$$

$$G_{pi}^{l+1} = G_p^{l+1} \text{ except that the } i\text{th row is replaced with zeros.} \tag{16}$$

$$\mathcal{G}_p^{l+1} = [G_p^{l+1} \quad aG_{p1}^{l+1} \cdots aG_{pp}^{l+1}] \in R^{p \times (p+1)(l+1)}. \tag{17}$$

c) We then use a recursive method to solve

$$\min_{W_p^{l+1}} ||\mathcal{Y}_p^{l+1} - (W_p^{l+1})^T \mathcal{G}_p^{l+1}||. \tag{18}$$

To obtain a recursive solution, we let

$$\begin{aligned} R_p^{l+1} &= \mathcal{G}_p^{l+1}(\mathcal{G}_p^{l+1})^T \\ &= R_p^l + k_p^{l+1}(k_p^{l+1})^T + a^2[k_{p1}^{l+1}(k_{p1}^{l+1})^T \\ &\quad + \cdots + k_{pp}^{l+1}(k_{pp}^{l+1})^T] \\ &= R_p^l + \sum_{i=0}^{p} \kappa_i \kappa_i^T \end{aligned} \tag{19}$$

where k_{pi}^{l+1} is the same as k_p^{l+1} except that the ith element is replaced with a zero. Recursively applying the matrix inversion lemma [6] to (19) gives the algorithm

$$(R_p^{l+1})^{-1} = (R_p^l)^{-1} \quad \text{for} \quad i = 0 \text{ to } p,$$

$$(R_p^{l+1})^{-1} = \frac{(R_p^{l+1})^{-1}\kappa_i \kappa_i^T (R_p^{l+1})^{-1}}{1 + \kappa_i^T (R_p^{l+1})^{-1}\kappa_i}. \tag{20}$$

Equation (18) is then solved as

$$W_p^{l+1} = (R_p^{l+1})^{-1}\mathcal{G}_p^{l+1}(\mathcal{Y}_p^{l+1})^T. \tag{21}$$

d) We next consider including z^{l+1} as a prototype. We will include z^{l+1} as a prototype only if we do not encounter ill conditioning in the solution of the problem

$$\min_{W_{p+1}^{l+1}} ||\mathcal{Y}_{p+1}^{l+1} - (W_{p+1}^{l+1})^T \mathcal{G}_{p+1}^{l+1}|| \tag{22}$$

where

$$\mathcal{G}_{p+1}^{l+1} = \begin{bmatrix} G_p^{l+1} & aG_{p1}^{l+1} & \cdots & aG_{pp}^{l+1} & aG_p^{l+1} \\ r_{l+1}^T & ar_{l+1}^T & \cdots & ar_{l+1}^T & \vec{0}^T \end{bmatrix} \tag{23}$$

$$r_{l+1}^T = [\phi(z^1, z^{l+1}) \ \cdots \ \phi(z^{l+1}, z^{l+1})]. \tag{24}$$

To determine if $\mathcal{G}_{p+1}^{l+1}(\mathcal{G}_{p+1}^{l+1})^T$ is well conditioned and to obtain a recursive solution to (22), we note that

$$\begin{aligned} R_{p+1}^{l+1} &= \mathcal{G}_{p+1}^{l+1}(\mathcal{G}_{p+1}^{l+1})^T \\ &= \begin{bmatrix} R_p^{l+1} + a^2 G_p^{l+1}(G_p^{l+1})^T & \mathcal{G}_p^{l+1}\hat{r}_{l+1} \\ \hat{r}_{l+1}^T(\mathcal{G}_p^{l+1})^T & \hat{r}_{l+1}^T\hat{r}_{l+1} \end{bmatrix} \end{aligned} \tag{25}$$

where the $(p+1)(l+1)$-element vector $\hat{r}_{l+1}$ is given by

$$\hat{r}_{l+1}^T = [r_{l+1}^T \quad ar_{l+1}^T \ \cdots \ ar_{l+1}^T]. \tag{26}$$

Recursively applying the matrix inversion lemma as in (20) results in

$$(R_{p+1}^{l+1})^{-1} = \begin{bmatrix} A^{-1} & \frac{-u}{\beta} \\ \frac{-u^T}{\beta} & \frac{1}{\beta} \end{bmatrix} \tag{27}$$

where A^{-1} is computed as

$$\begin{aligned} A^{-1} &= (R_p^{l+1})^{-1} \\ &\text{for} \quad i = 1 \text{ to } l+1, \\ &A^{-1} = \frac{A^{-1}\mathbf{g}_i\mathbf{g}_i^T A^{-1}}{1 + \mathbf{g}_i^T A^{-1}\mathbf{g}_i}. \end{aligned} \tag{28}$$

The vector $\mathbf{g}_i$ is defined as the ith column in G_p^{l+1}, and β and u are given by

$$\beta = \hat{r}^T[I - (\mathcal{G}_p^{l+1})^T A^{-1}\mathcal{G}_p^{l+1}]\hat{r} \tag{29}$$

$$u = \frac{A^{-1}\mathcal{G}_p^{l+1}\hat{r}_{l+1}}{\beta}. \tag{30}$$

e) We monitor the value of β to prevent ill conditioning. A threshold γ is chosen such that z^{l+1} is included as a prototype only if $\beta > \gamma$. If in fact β is greater than γ, we compute $(R_{p+1}^{l+1})^{-1}$ using (27)–(30), form the $m \times (l+1)(p+2)$ matrix

$$\mathcal{Y}_{p+1}^{l+1} = [\mathcal{Y}_p^{l+1} \quad aY_{l+1}] \tag{31}$$

and solve (22) as

$$W_{p+1}^{l+1} = (R_{p+1}^{l+1})^{-1}\mathcal{G}_{p+1}^{l+1}(\mathcal{Y}_{p+1}^{l+1})^T. \tag{32}$$

We then augment the subprototype z^{l+1} to the prototype matrix V and the subprototype matrix Z, and increment p and l by one.

f) If $\beta < \gamma$ then z^{l+1} cannot be included as a prototype. We augment z^{l+1} to the subprototype matrix Z and increment l by one to reflect the addition of a new subprototype.

3) Reiterate.

After Step 2) we check if any new subprototypes were added to Z. If so, then the network has been modified, and we have to check if the exemplars remaining in A can still be correctly classified. So we set $n = q - l$, reindex the exemplars in A from one to n, reindex the corresponding outputs, and go back to Step 2).

If no new subprototypes were added during Step 2), then the learning procedure terminates. It is clear that Step 2) is executed $q - 1$ times at the most [3].

In the basic OI net learning algorithm [3] a recursive computation of the error was derived. A given exemplar was included as a prototype only if the resultant decrease in classification error was large enough to justify the associated increase in variance. When fault tolerance is added to the learning algorithm as presented in this section, however, there is no apparent way to recursively compute the classification error. Of course, a user can still compute the error decrease to ensure that an exemplar is worth adding as a prototype. But since there is no recursive method available for this computation, it has not been included in the algorithm presented in this section.

IV. Simulation Results

The fault-tolerant OI net discussed in this paper was applied to the problem of navigation satellite selection. Comparison of the OI net, backpropagation, and nearest-neighbor classification has previously been presented [3]. So the data in this section are limited to OI net results.

A. Navigation Satellite Subset Selection

A global positioning system (GPS) receiver generates a user position and time by measuring the range from the user to four or more GPS satellites [7]–[8], but a GPS receiver can process only a subset of available satellite signals. So before processing, the receiver must decide which subset to use. The optimal choice can be made by using the subset which results in the smallest magnification of satellite errors onto resultant user position and time.

A user's GPS receiver measures a set of n ranges $(R_1, R_2, \cdots, R_n)$ between the user and n GPS satellites. The GPS satellites are at positions (x_i, y_i, z_i), $(i = 1, \cdots, n)$. The four unknowns which the user needs to determine are the offset T between receiver time and GPS time, and the user position (x, y, z). We denote the user's best estimate of time offset and position as $\hat{T}$ and $(\hat{x}, \hat{y}, \hat{z})$. We denote the corresponding best estimates of range as $(\hat{R}_1, \hat{R}_2, \cdots, \hat{R}_n)$.

The errors between the true and estimated quantities are denoted by Δx, Δy, Δz, ΔT, and ΔR_i. The errors of the user's estimate of time and position can be determined by solving the following n simultaneous nonlinear equations for Δx, Δy, Δz, and ΔT [9]

$$(\hat{x}+\Delta x-x_i)^2+(\hat{y}+\Delta y-y_i)^2+(\hat{z}+\Delta z-z_i)^2= (\hat{R}_i+\Delta R_i-c\hat{T}-c\Delta T)^2 \quad (i=1,\cdots,n) \tag{33}$$

where c is the speed of light. These equations can be linearized to obtain the equation

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 1 \\ a_{21} & a_{22} & a_{23} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & 1 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ c\Delta T \end{pmatrix} = \begin{pmatrix} \Delta R_1 - c\hat{T} \\ \Delta R_2 - c\hat{T} \\ \vdots \\ \Delta R_n - c\hat{T} \end{pmatrix} \Longrightarrow A\vec{x}=\vec{r}. \tag{34}$$

If the covariance of $\vec{r}$ is normalized to an identity matrix, we obtain a simplified expression for the covariance of user position and time

$$\text{cov}(\vec{r})=I \Longrightarrow \text{cov}\,(\vec{x})=(A^TA)^{-1}. \tag{35}$$

A useful scalar measure of the magnification of GPS range measurement errors onto user position and time errors is the square root of the trace of the above matrix. This quantity is referred as geometric dilution of precision (GDOP)

$$\text{GDOP}=\sqrt{\text{trace}\,(A^TA)^{-1}}. \tag{36}$$

How can GDOP be computed without resorting to matrix inversion? Recall the following general facts about the trace and eigenvalues of a matrix [10], [11]:

1) The trace of a matrix is equal to the sum of its eigenvalues,
2) The determinant of a matrix is equal to the product of its eigenvalues, and
3) If A has eigenvalues λ_i then A^k has eigenvalues λ_i^k, where k is any integer.

Using $\vec{\lambda}$ to denote the four-element vector of the eigenvalues of A^TA, we can define the following four functions

$$f_1(\vec{\lambda})=\lambda_1+\lambda_2+\lambda_3+\lambda_4=\text{trace}\,(A^TA) \tag{37}$$
$$f_2(\vec{\lambda})=\lambda_1^2+\lambda_2^2+\lambda_3^2+\lambda_4^2=\text{trace}\,[(A^TA)^2] \tag{38}$$
$$f_3(\vec{\lambda})=\lambda_1^3+\lambda_2^3+\lambda_3^3+\lambda_4^3=\text{trace}\,[(A^TA)^3] \tag{39}$$
$$f_4(\vec{\lambda})=\lambda_1\lambda_2\lambda_3\lambda_4=\det\,(A^TA). \tag{40}$$

Using the above notation, the GDOP which we wish to calculate is given as a scalar functional of the $\mathcal{R}^4 \rightarrow \mathcal{R}^4$ mapping $\vec{f}(\vec{\lambda})$

$$\text{GDOP}=\sqrt{\lambda_1^{-1}+\lambda_2^{-1}+\lambda_3^{-1}+\lambda_4^{-1}} =\text{GDOP}[\vec{f}(\vec{\lambda})]. \tag{41}$$

The mapping from $\vec{f}(\vec{\lambda})$ to GDOP cannot be determined analytically. But this complex, nonlinear mapping is the type of problem at which neural networks excel. A neural network can be designed to inductively generate a GDOP classification algorithm by generalizing from known input–output relationships [12], [13].

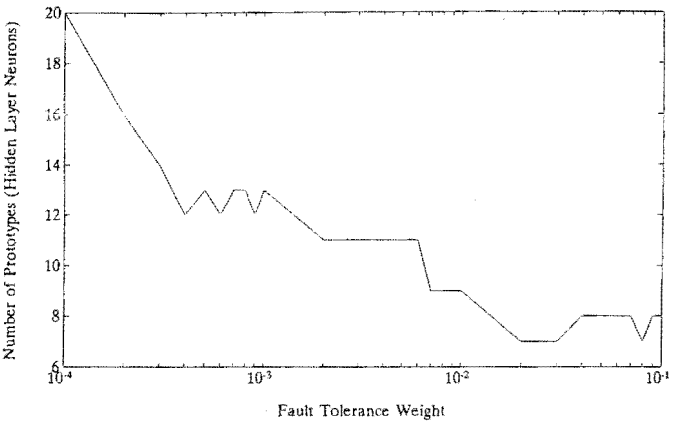

Fig. 1. Effect of fault tolerance weight on number of neurons.

V. Results

The fault-tolerant OI net described in this paper was simulated on a VAX 8650 computer. Training took place for a GPS receiver located at 5000 feet above San Francisco (37.5 degrees latitude, 122 degrees longitude) in an 18-satellite constellation. Once each hour, for 12 hours, the functions f_i $(i=1,\ 2,\ 3,\ 4)$ were calculated for each visible four-satellite subset, and GDOP was calculated by explicitly inverting A^TA. If f_4 was less then 0.12, the satellite set was immediately discarded from consideration. Such a low determinant can be shown by simulation to correspond to a GDOP too high for consideration. At each training time there were between five and seven visible satellites. There were thus between 15–35 four-satellite sets from which to choose.

The network was then tested on a simulated 120-second missile trajectory. The trained neural network was used to classify each satellite group (according to GDOP) every two seconds. There were between five and seven satellites visible during the boost phase, and the satellite configuration with the best GDOP changed twice during that time.

Several OI nets were trained and tested for different values of the weight a. Each OI net had three input neurons corresponding to f_2, f_3, and f_4 (since f_1 was constant), and each OI net had two output neurons. A satellite group with a GDOP less than the classification threshold should have an output vector of $[1,\ 0]$, and a group with a GDOP greater than the threshold should have an output vector of $[0,\ 1]$. Each OI net used a fitting parameter $\rho=0.1$ and an ill-conditioning threshold $\gamma=10^{-8}$. Fig. 1 shows the number of prototypes (hidden layer neurons) generated as a function of the weight a. In general, the number of prototypes decreases as a increases. This is because we have assumed that each hidden layer neuron has a fixed probability of failure, so the probability of a network failure increases linearly with the number of hidden layer neurons. Fig. 1 reflects the fact that a smaller network has a smaller probability of failure. If we wish

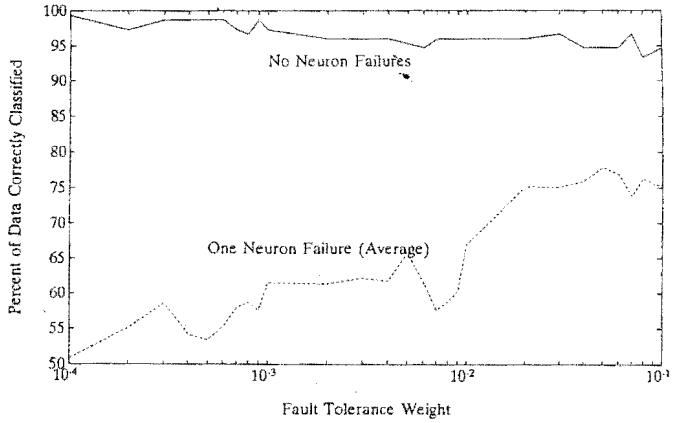

Fig. 2. Classification performance on training data.

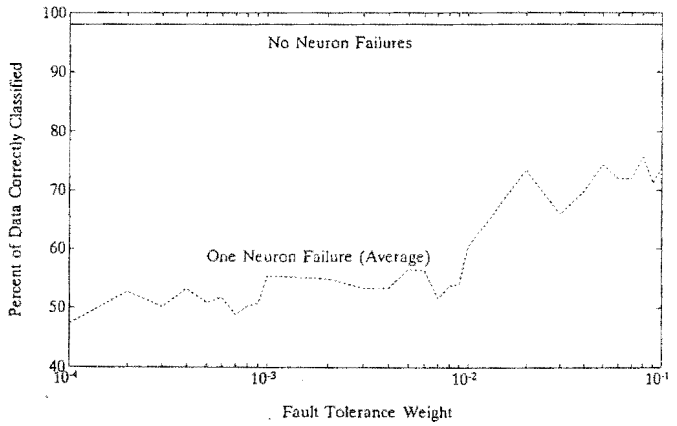

Fig. 3. Classification performance on test data.

to assume that a network failure is equally likely regardless of the number of neurons, we could replace a in (8) with a/p. This change can be reflected in the learning algorithm of Section III in a straightforward manner.

Figs. 2 and 3 show the performance of the fault-tolerant OI net for various values of a. It is seen that increasing a significantly improves the fault tolerance of the OI net. It is also seen that increasing a does not hurt the performance of the net even if there are no neuron failures. At first glance this result is surprising. But adding fault tolerance can be thought of as protecting the net against misclassification due to noisy data. This leads to improved generalization properties and good performance even for the nominal network. This shows that even if we consider the probability of failure negligible, it pays to build fault tolerance into the network.

Of course, we cannot get something for nothing. The price we pay for fault tolerance is increased learning time. The addition of fault tolerance results in an increase by a factor of p of the size of many of the matrices in the learning algorithm of Section III. But the recursive learning algorithm is so efficient that this increase in training time is probably not a critical factor. Table II shows the increase in training time due to the introduction of fault tolerance.

TABLE II
OI NET TRAINING TIMES IN VAX CPU SECONDS (150 TRAINING INPUTS)

	Fault Tolerance Weight	Number of Prototypes	Training Time
Basic OI Net	-	15	1.22
Fault Tolerant OI Nets	0.000	15	5.95
	0.001	13	4.55
	0.005	11	3.46
	0.010	9	2.82
	0.020	7	2.46

VI. CONCLUSION

A recursive learning algorithm for a fault-tolerant OI net has been presented. The inclusion of fault tolerance increases the training time by a factor of between two and five, depending on the weight given to fault tolerance. But fault tolerance improves the generalization properties of the network while at the same time decreasing the number of hidden layer neurons (and hence decreasing the complexity of the network).

The fault tolerance discussed in this paper applies to a single neuron failure. An extension to tolerance for failures of two or more neurons is conceptually straightforward, but may give rise to large increases in training time.

The fault-tolerant OI net has been applied to the navigation satellite selection problem. The simulated results show that not only is fault tolerance increased, but nominal performance does not suffer relative to an OI net without fault tolerance. This is because the introduction of fault tolerance can be viewed as protecting the network against noisy data, and hence improving the generalization properties of the network.

REFERENCES

[1] E. Karnin, "A simple procedure for pruning backpropagation trained neural networks," *IEEE Trans. Neural Networks,* vol. 1, pp. 239–242, June 1990.
[2] R. deFigueiredo, "An optimal matching-score net for pattern classification," in *Proc. IJCNN,* vol. III, San Diego, CA, June 17–21, 1990, pp. 909–916.
[3] S. Sin and R. deFigueiredo, "An evolution-oriented learning algorithm for the optimal interpolative net," *IEEE Trans. Neural Networks,* vol. 3, no. 2, pp. 315–323, 1992.
[4] ——, "Efficient learning procedures for optimal interpolative nets," *Neural Networks,* vol. 6, pp. 99–113, 1993.
[5] C. Neti, M. Schneider, and E. Young, "Maximally fault tolerant neural networks," *IEEE Trans. Neural Networks,* vol. 3, pp. 14–23, Jan. 1992.
[6] L. Ljung and T. Soderstrom, *Theory and Practice of Recursive Identification.* Cambridge, MA: MIT Press, 1985.
[7] H. El-Sherief, "Aerospace applications of global positioning system," in *Proc. Electrical Engineering Seminar,* Univ. Nevada, Reno, 1992.
[8] P. Janiczek and S. Gilbert, Eds., *Global Positioning Syst.,* vols. 1–3. Washington, DC: Institute Navigation, 1980, 1984, 1986.
[9] P. Jorgensen, "Navstar/global positioning system 18-satellite constellations," in *Global Positioning Syst.,* vol. 2. Washington, DC: Institute Navigation, 1984, pp. 1–12.
[10] G. Golub and C. Van Loan, *Matrix Computation,* 2nd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1990.
[11] R. Horn and C. Johnson, *Matrix Analysis.* New York, NY: Cambridge Univ. Press, 1990.
[12] C. Townsend and D. Feucht, *Designing and Programming Personal Expert Systems.* Blue Ridge Summit, PA: Tab Books, 1986.
[13] D. Simon and H. El-Sherief, "A fault tolerant optimal interpolative net," in *Proc. IEEE Conf. Neural Networks,* vol. II, San Francisco, CA, Mar. 28–Apr. 1, 1993, pp. 825–830.