D.C. Park, M. El-Sharkawi and R.J. Marks II, "An adaptively trained neural network", IEEE Transactions on Neural Networks, vol.2, pp.334-345, (1991).

# An Adaptively Trained Neural Network

Dong C. Park, *Member, IEEE*, Mohamed A. El-Sharkawi, *Senior Member, IEEE*, and Robert J. Marks II, *Senior Member, IEEE*

***Abstract*—A training procedure is proposed that adapts the weights of a trained layered perceptron artificial neural network to training data originating from a slowly varying nonstationary process. The resulting adaptively trained neural network (ATNN), based on nonlinear programming techniques, is shown to adapt to new training data that is in conflict with earlier training data without affecting the neural networks' response to data elsewhere. The adaptive training procedure also allows for new data to be weighted in terms of its significance. The adaptive algorithm is applied to the problem of electric load forecasting and is shown to significantly outperform the conventionally trained layered perceptron.**

## I. Introduction

In the training of a layered perceptron, an assumption of stationarity of the training data is typically made. In a number of cases of interest, however, the training data constitute a slowly varying nonstationary process. In this paper, we present a training procedure applicable to such cases. In order for the layered perceptron's weights to adapt to a slowly varying nonstationarity, such a procedure should 1) still respond appropriately to previous training data if those data are not in conflict with the new training data and 2) adapt to the new training data even when they are in conflict with portions of the old data.

We propose a procedure for such adaptation which is applicable when the training data's stationarity varies sufficiently slowly. Our procedure for adaptive updating ensures a proper response to previous training data by seeking to minimize a weight sensitivity cost function while minimizing the mean square error normally ascribed to the layered perceptron. The process is illustrated through application to an examplar interpolation problem and through its use on an electric load forecasting problem with data generated by the power industry.

## II. Formulation of the Problem

Assume a layered perceptron artificial neural network trained with $N$ sets of data,

$$\{(\boldsymbol{x}(1), d(1)), (\boldsymbol{x}(2), d(2)), \cdots, (\boldsymbol{x}(N), d(N))\}$$

where $\boldsymbol{x}(i)$ and $d(i)$ represent the input and desired output for the $i$th data set and $1 \le i \le N$. We assume that $\boldsymbol{x}(i)$ is an $I$-dimensional vector and $d(i)$ is scalar. The layered perceptron is assumed to have one hidden layer with $h$ hidden neurons. The case for more than one hidden layer is given in Appendix I. The matrix $\boldsymbol{W}$ represents the weight matrix between the input and hidden neurons and $\boldsymbol{v}$ denotes the weight vector which links the hidden and output neurons. The dimensions of $\boldsymbol{W}$ and $\boldsymbol{v}$ are $I \times h$ and $h \times 1$, respectively.

For a given input data vector, $\boldsymbol{x}(i)$, the output of the layered perceptron of $y(i)$ is given by

$$y(i) = f[\boldsymbol{v}^T \boldsymbol{u}] \tag{1}$$

$$\boldsymbol{u} = \boldsymbol{f}[\boldsymbol{W}^T \boldsymbol{x}(i)] \tag{2}$$

where $\boldsymbol{u} = [u_1, u_2, \cdots, u_h]^T$; $u_j$, $1 \le j \le h$, represents the activation of the $j$th hidden neuron; the superscript $T$ denotes the transpose of a matrix or vector; $f[\cdot]$ is the sigmoid function; $f[x] = 1/(1 + \exp(-x))$, $x \in \Re$; and $\boldsymbol{f}[\boldsymbol{b}]$ is the $h \times 1$ vector function

$$\boldsymbol{f}[\boldsymbol{b}] = [f_1[\boldsymbol{b}], f_2[\boldsymbol{b}], \cdots, f_h[\boldsymbol{b}]]^T.$$

The sigmoid function for each of the hidden neurons is assumed to be identical, $f_1[\cdot] = f_2[\cdot] = \cdots = f_h[\cdot] = f[\cdot]$. We assume that $\boldsymbol{W}(N)$ and $\boldsymbol{v}(N)$ are the weights that minimize the error function [1]:

$$E(N) = \tfrac{1}{2} \sum_{i=1}^{N} (d(i) - y(i))^2. \tag{3}$$

## III. Problem Statement

The statement of our problem can now be described as follows:

Given $\boldsymbol{W}(N)$, $\boldsymbol{v}(N)$, the $N$ sets of data, and $(\boldsymbol{x}(N+1), d(N+1))$, determine $\boldsymbol{W}(N+1)$ and $\boldsymbol{v}(N+1)$ such that

$$E(N+1) = \tfrac{1}{2} \sum_{i=1}^{N+1} (d(i) - y(i))^2$$

$$= E(N) + \tfrac{1}{2}(d(N+1) - y(N+1))^2 \tag{4}$$

is minimized in such a manner that $y(N+1) \approx d(N+1)$.

## IV. Linearization Process

Linearization of the problem allows for analytic and implementation tractibility: For $(\boldsymbol{x}(N+1), d(N+1))$, (1) and (2) yield

$$d(N+1) = f[\boldsymbol{v}^T(N+1)\boldsymbol{u}] = f[(\boldsymbol{v}(N) + \Delta\boldsymbol{v})^T \boldsymbol{u}] \tag{5}$$

Manuscript received May 7, 1990; revised December 12, 1990. This work was supported by the Puget Sound Power and Light Company and by the Washington Technology Center at the University of Washington.

D. C. Park is with the Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33199.

M. A. El-Sharkawi and R. J. Marks II are with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195.

IEEE Log Number 9143023.

1045-9227/91/0500-0334$01.00 © 1991 IEEE

where

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{f}[\boldsymbol{W}^T(N+1)\,\boldsymbol{x}(N+1)] \\ &= \boldsymbol{f}[(\boldsymbol{W}(N)+\Delta\boldsymbol{W})^T\boldsymbol{x}(N+1)]. \end{aligned} \tag{6}$$

We expand (5) and (6) in a truncated Taylor series about the state of interest, $\{\boldsymbol{x}(N+1), \boldsymbol{W}(N+1)\}$, that is closest to the current operating state, $\{\boldsymbol{x}(N), \boldsymbol{W}(N)\}$. Such a linearization is used in other adaptive filter techniques, such as the extended Kalman filtering [2], [3] and quasi linearization [4]–[6].

Let

$$\boldsymbol{b} = \boldsymbol{W}^T(N)\,\boldsymbol{x}(N+1)$$

and

$$\Delta\boldsymbol{b} = \Delta\boldsymbol{W}^T\boldsymbol{x}(N+1),$$

Application of a first-order Taylor series expansion to (6) gives

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{f}[\boldsymbol{W}^T(N)\,\boldsymbol{x}(N+1) + \Delta\boldsymbol{W}^T\boldsymbol{x}(N+1)] \\ &= \boldsymbol{f}[\boldsymbol{b}+\Delta\boldsymbol{b}] \\ &\approx \boldsymbol{f}[\boldsymbol{b}] + (\nabla_b\boldsymbol{f}[\boldsymbol{b}])\Delta\boldsymbol{b} \end{aligned}$$

where $\nabla_b\boldsymbol{f}[\boldsymbol{b}]$ is the gradient of $\boldsymbol{f}[\boldsymbol{b}]$ with respect to $\boldsymbol{b}$ with elements $\partial f_i[\boldsymbol{b}]/\partial b_j$. This approximation is valid when $H \ll 1$, where $H$ is the Hessian of $\boldsymbol{f}[\boldsymbol{b}]$. We require that the perturbation, $\Delta\boldsymbol{b}$ (or $\Delta\boldsymbol{W}$), be small enough so that

$$\Delta\boldsymbol{b} \ll \boldsymbol{f}[\boldsymbol{b}]/H$$

where the inequality applies to each component of the vector.

Since $f_i[\,\cdot\,] = \boldsymbol{f}[\,\cdot\,]$, the $i$th component of $\boldsymbol{f}[\boldsymbol{b}]$ is only a function of $b_i$ (that is, $f_i[\boldsymbol{b}] = f_i[b_i]$, $1 \le i \le h$) and

$$\begin{aligned} \partial f_i[\boldsymbol{b}]/\partial b_j &= \partial f[b_i]/\partial b_j \\ &= f[b_i]\,(1 - f[b_i])\,\delta_{i-j} \\ &= u_i^*(1-u_i^*)\,\delta_{i-j} \end{aligned}$$

then,

$$\nabla_b\boldsymbol{u}^* = \text{diag}\,[u_1^*(1-u_1^*),\ u_2^*(1-u_2^*),\ \cdots,\ u_h^*(1-u_h^*)]$$

where $\delta_k$, the Kronecker delta, is 1 for $k = 0$ and is zero otherwise. Also, $u_i^* = f[b_i]$ is the activation of the $i$th hidden neuron for the new input data with the old weight such as

$$\boldsymbol{u}^* = [u_1^*, u_2^*, \cdots, u_h^*]^T = \boldsymbol{f}[\boldsymbol{W}^T(N)\,\boldsymbol{x}(N+1)].$$

Therefore, the activations of hidden neurons are given by

$$\boldsymbol{u} \simeq \boldsymbol{u}^* + (\nabla_b\boldsymbol{u}^*)\Delta\boldsymbol{W}^T\boldsymbol{x}(N+1). \tag{7}$$

Inverting (5) yields

$$\boldsymbol{f}^{-1}[d(N+1)] = \boldsymbol{v}^T(N)\boldsymbol{u} + \Delta\boldsymbol{v}^T\boldsymbol{u} \tag{8}$$

where $\boldsymbol{f}^{-1}[x] = \ln(x/1-x))$.

From (7) and (8) we find that

$$\begin{aligned} &\boldsymbol{f}^{-1}[d(N+1)] - \boldsymbol{v}^T(N)\boldsymbol{u}^* \\ &\quad\simeq \boldsymbol{v}^T(N)\,(\nabla_b\boldsymbol{u}^*)\,\Delta\boldsymbol{W}^T\boldsymbol{x}(N+1) \\ &\qquad + \Delta\boldsymbol{v}^T\boldsymbol{u}^* + \Delta\boldsymbol{v}^T(\nabla_b\boldsymbol{u}^*)\,\Delta\boldsymbol{W}^T\boldsymbol{x}(N+1). \end{aligned} \tag{9}$$

Remember that the perturbation in $\Delta\boldsymbol{W}$ is assumed to be small in order to use (7). If $\Delta\boldsymbol{v} \ll \boldsymbol{v}$, then the third term of the right-hand side of (9) can be ignored and

$$\begin{aligned} &\boldsymbol{f}^{-1}[d(N+1)] - \boldsymbol{v}^T(N)\boldsymbol{u}^* \\ &\quad\simeq \boldsymbol{v}^T(N)\,(\nabla_b\boldsymbol{u}^*)\,\Delta\boldsymbol{W}^T\boldsymbol{x}(N+1) + \Delta\boldsymbol{v}^T\boldsymbol{u}^*. \end{aligned} \tag{10}$$

We rearrange $\Delta\boldsymbol{W}$ into vector form as follows:

$$\Delta\boldsymbol{W}_{\text{vec}} = [\Delta\boldsymbol{w}_1^T \cdots \Delta\boldsymbol{w}_I^T]^T = [\Delta W_{\text{vec},1} \cdots \Delta W_{\text{vec},p}]$$

where $\Delta\boldsymbol{w}_i$ is the $i$th row in $\Delta\boldsymbol{W}$, and $p = h \times I$. Note that $p$ is the number of interconnections between the neurons in the input and the hidden layers. Then (10) can be rewritten as

$$\begin{aligned} c_1 &= [\Delta\boldsymbol{W}_{\text{vec}}^T:\ \Delta\boldsymbol{v}^T]\begin{bmatrix}\boldsymbol{u}^\dagger \\ \cdot\cdot \\ \boldsymbol{u}^*\end{bmatrix} \\ &= \boldsymbol{z}^T\boldsymbol{a} \end{aligned} \tag{11}$$

where $c_1$, $\boldsymbol{a}$, and $\boldsymbol{z}$ are vectors defined as

$$c_1 \equiv \boldsymbol{f}^{-1}[d(N+1)] - \boldsymbol{v}^T(N)\boldsymbol{u}^* \tag{12}$$

$$\boldsymbol{a} \equiv [\boldsymbol{u}^{\dagger T}:\ \boldsymbol{u}^{*T}]^T \tag{13}$$

$$\boldsymbol{z} \equiv [\Delta\boldsymbol{W}_{\text{vec}}^T:\ \Delta\boldsymbol{v}^T]^T \tag{14}$$

and $\boldsymbol{u}^\dagger$ is a solution of

$$\Delta\boldsymbol{W}_{\text{vec}}^T\boldsymbol{u}^\dagger = \boldsymbol{v}^T(N)\,\boldsymbol{Q}\Delta\boldsymbol{W}^T\boldsymbol{x}(N+1). \tag{15}$$

Since there are $h \times (I+1)$ unknowns with one equation, (11), there exist many solutions. Uniqueness is imposed by the additional requirement that our solution be constrained by the additional conditions in the problem statement. Specifically, we need to change the weights for the $(N+1)$st datum with minimum effect on the previous $N$ data. We are therefore motivated to find the sensitivity of $y(i)$ over a weight change. Equation (4) can be rewritten as

$$E(N) = \tfrac{1}{2}\sum_{i=1}^{N}(d(i) - y(i))^2 = \sum_{i=1}^{N} E_i.$$

The sensitivity for the input weights follows as

$$\frac{\partial E_i}{\partial w_{jk}} = -[d(i) - y(i)]\left(\frac{\partial y(i)}{\partial w_{jk}}\right) \tag{16}$$

and those for the output weights are

$$\frac{\partial E_i}{\partial v_k} = -[d(i) - y(i)]\left(\frac{\partial y(i)}{\partial v_k}\right) \tag{17}$$

where $w_{jk}$, the weight of interconnection between input neuron $j$ and hidden neuron $k$, is the $jk$th element of $\boldsymbol{W}$.

Let

$$y(i) = f[\text{sum}_y] \quad \text{and} \quad u_k = f[\text{sum}_u]$$

where

$$\text{sum}_y = \sum_k u_k v_k \quad \text{and} \quad \text{sum}_u = \sum_j w_{jk} x_j.$$

Thus,

$$\begin{aligned}\frac{\partial y(i)}{\partial v_k} &= \left(\frac{\partial y(i)}{\partial \text{sum}_y}\right)\left(\frac{\partial \text{sum}_y}{\partial v_k}\right) = \left.\frac{\partial f[x]}{\partial x}\right|_{x=y(i)} u_k \\ &= y(i)\,(1 - y(i))u_k \\ &\equiv SV_{i,k} \end{aligned} \tag{18}$$

and

$$\begin{aligned}\frac{\partial y(i)}{\partial w_{jk}} &= \left(\frac{\partial y(i)}{\partial u_k}\right)\left(\frac{\partial u_k}{\partial w_{jk}}\right) \\ &= \left(\frac{\partial y(i)}{\partial \text{sum}_y}\right)\left(\frac{\partial \text{sum}_y}{\partial u_k}\right)\left(\frac{\partial u_k}{\partial \text{sum}_u}\right)\left(\frac{\partial \text{sum}_u}{\partial w_{jk}}\right) \\ &= \left.\frac{\partial f[x]}{\partial x}\right|_{x=y(i)} v_k \left.\frac{\partial f[x]}{\partial x}\right|_{x=u_k} x_j \\ &= y(i)\,(1 - y(i))\,v_k u_k (1 - u_k) x_j \\ &\equiv SW_{i,jk} \end{aligned} \tag{19}$$

where $SV_{i,k}$ is the sensitivity of $y(i)$ caused by small changes in $v_k$ and $SW_{i,jk}$ is the sensitivity of $y(i)$ to the $w_{j,k}$'s. Consequently,

$$\Delta E_i = \sum_{j,k}\left(\frac{\partial E_i}{\partial w_{jk}}\right)\Delta \boldsymbol{W}_{jk} + \sum_k \left(\frac{\partial E_i}{\partial v_k}\right)\Delta \boldsymbol{v}_k. \tag{20}$$

By combining (18), (19), and (20) with (16) and (17), we obtain

$$\begin{aligned}\Delta E_i = (\epsilon_i)\,[&\Delta W_{\text{vec},1} \cdots \Delta W_{\text{vec},p}: \Delta v_1 \cdots \Delta v_h] \\ &\cdot [SW_{i,1} \cdots SW_{i,p} \cdots : SV_{i,1} \cdots SV_{i,h}]^T \end{aligned} \tag{21}$$

where $\epsilon_i = -(d(i) - y(i))$, $1 \le i \le N$, and $p = I \times h$. Equivalently, using (20), we can write the matrix equation

$$\Delta \boldsymbol{E} = \boldsymbol{A}\boldsymbol{S}\boldsymbol{z} \tag{22}$$

where the $i$th element of $\Delta \boldsymbol{E}$ is $\Delta E_i$:

$$\boldsymbol{A} = \text{diag}\,[\epsilon_1, \cdots, \epsilon_N] \tag{23}$$

and

$$\boldsymbol{S} = \begin{bmatrix} SW_{1,1} & \cdots & SW_{1,p} & SV_{1,1} & \cdots & SV_{1,h} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ SW_{N,1} & \cdots & SW_{N,p} & SV_{N,1} & \cdots & SV_{N,h} \end{bmatrix} \tag{24}$$

where $\boldsymbol{z}$ is a $q \times 1$ vector defined in (11) and $q = p + h = (I + 1) \times h$.

The weights $\boldsymbol{W}(N)$ and $\boldsymbol{v}(N)$ are optimal for minimizing $E(N)$. With the addition of the $(N + 1)$st datum, $d(N + 1) = y(N + 1)$, the objective function of (4) can be changed to

$$\begin{aligned} J &= \tfrac{1}{2}\sum_{i=1}^{N}\left(E_{i,W(N)} - E_{i,W(N+1)}\right)^2 \\ &= \tfrac{1}{2}\sum_{i=1}^{N}\Delta E_i^2 \end{aligned}$$

where $E_{i,W(N)}$ and $E_{i,W(N+1)}$ are the errors of the $i$th datum with $\{\boldsymbol{W}(N), \boldsymbol{v}(N)\}$ and $\{\boldsymbol{W}(N + 1), \boldsymbol{v}(N + 1)\}$, respectively. Equivalently, from (22),

$$\begin{aligned} J &= \tfrac{1}{2}(\Delta \boldsymbol{E})^T(\Delta \boldsymbol{E}) \\ &= \tfrac{1}{2}(\boldsymbol{A}\boldsymbol{S}\boldsymbol{z})^T(\boldsymbol{A}\boldsymbol{S}\boldsymbol{z}) \\ &= \tfrac{1}{2}\boldsymbol{z}^T\boldsymbol{K}\boldsymbol{z} \end{aligned} \tag{25}$$

where $\boldsymbol{K} = \boldsymbol{S}^T(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{S}$. Note that $\boldsymbol{K} = \boldsymbol{K}^T$.

Since there exists only one equation with $q$ unknowns in (11), the solutions of (11) are on a $(q - 1)$-dimensional hyperplane. Since small perturbations for $\boldsymbol{W}$ and $\boldsymbol{v}$ are assumed in (7), among the solutions in (11), only those with small variation in weight space (i.e., small $\|\boldsymbol{z}\|$) are allowable.

The problem is now a standard nonlinear programming problem. Specifically,

$$\begin{aligned} &\text{minimize} && J(\boldsymbol{z}) = \tfrac{1}{2}\boldsymbol{z}^T\boldsymbol{K}\boldsymbol{z} \\ &\text{subject to} && \boldsymbol{z}^T\boldsymbol{a} = c_1. \end{aligned}$$

A equicost line of the cost function $J(\boldsymbol{z})$ represents a $q$-dimensional hyperellipse centered at the origin of the $\boldsymbol{z}$ plane. The constraint $\boldsymbol{z}^T\boldsymbol{a} = c_1$ is a $(q - 1)$-dimensional hyperplane on the $\boldsymbol{z}$ plane. The shape of the cost function is dictated by the eigenvalues and eigenvectors of $\boldsymbol{K}$ [7], [8].

When there is no boundary constraint or too loose of a boundary constraint for the value of $\boldsymbol{z}$, the choice of $\boldsymbol{z}$ (say $\boldsymbol{z}^*$) may minimize the cost index $J(\boldsymbol{z})$, at the cost of severely violating the linearity assumption. If we introduce a boundary constraint, $\mathcal{B}$, for $\boldsymbol{z}$, we can avoid this problem at a cost of being at a low though not minimum value of the cost function. However, if we use too restrictive (or tight) a boundary, as shown in Fig. 1, a solution that satisfies the two different constraints may not exist.

The above discussion suggests that we need to limit the perturbations to make them small enough to meet the linearization assumption of the truncated Taylor series. At the same time, we need to allow $\boldsymbol{z}$ to be large enough to have at least one solution on the $\boldsymbol{z}^T\boldsymbol{a} = c_1$ plane.

## V. Boundary Constraint

Define the boundary as a set of points in the $\boldsymbol{z}$ plane:

$$\mathcal{B} = \{\boldsymbol{z}: -\vec{\ell} \le \boldsymbol{z} \le \vec{\ell}\}$$

where $\vec{\ell} = [l_1, l_2, \cdots, l_q]^T$ and $l_i \ge 0$, $i = 1, \cdots, q$.

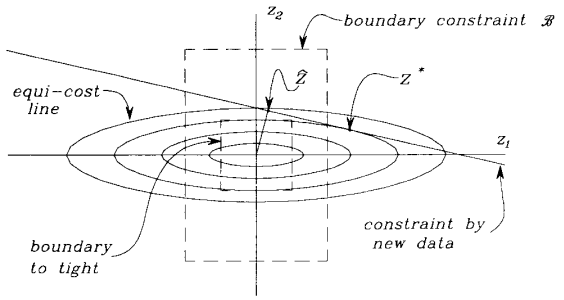



Fig. 1. Contours of constant $J(z)$ when $z$ is of dimension 2. The constraint imposed by the new data is shown as a line.

The $(q-1)$-dimensional constraint hyperplane formed by $z^T a = c_1$ contains a unique point, $\hat{z}$, that is minimal distance from the origin. The boundary constraint should include at least $\hat{z}$. In other words, we need to choose $\vec{\ell}$ that includes at least $\hat{z}$. For the plane $z^T a = c_1$, the result is

$$\hat{z} = \frac{c_1 a}{a^T a}. \tag{26}$$

By using $\hat{z}$ given in (26), we choose $\vec{\ell}$ such as $\vec{\ell} = k\hat{z}$, $k \geq 1$. This boundary ensures intersection by the linear constraint plane. A minimal distance point and an example of a properly selected boundary constraint for the two-dimensional $z$ are shown in Fig. 1. The larger the $k$, the smaller $J(z)$ can be. Choice of an overly large value for $k$, however, may violate the Taylor series approximation used in (7).

## VI. Reduced Gradient Method

Using the boundary constraint, our problem now becomes

$$\begin{aligned} \text{minimize} \quad & J(z) = \tfrac{1}{2} z^T K z \\ \text{subject to} \quad & z^T a = c_1 \\ \text{and} \quad & z \in \mathcal{B}, \quad \text{or} \quad -\vec{\ell} \leq z \leq \vec{\ell}. \end{aligned} \tag{27}$$

There exists a variety of nonlinear programming methods to solve this type of problem (e.g., the gradient projection method and the convex simplex method [9]–[11]). We found the *reduced gradient method* (RGM) using a linear constraint most conducive to the problem.

The RGM starts with the partitioning of variables into two groups: basic and nonbasic. The variables in the basic group are dependent and the variables in the nonbasic group are independent. The cardinality of the basic group is the number of linear constraints. In our case, we will have one basic variable since there is one given linear constraint. The other $(q-1)$ variables are assigned as independent variables.

Assume that we have a feasible solution, $z$, which satisfies the constraints in (27). We partition the $q$-dimensional variable $z$ arbitrarily into two groups: $z = (z_\alpha, z_\beta) = [z_\alpha, z_\beta^T]^T$, where $z_\alpha$ is a one-dimensional basic (dependent) variable and $z_\beta$ is a $(q-1)$-dimensional nonbasic (independent) variable. The problem in (27) can then be written as

$$\text{minimize} \quad J(z_\alpha, z_\beta) = \tfrac{1}{2}(z_\alpha, z_\beta)^T K (z_\alpha, z_\beta) \tag{28}$$

$$\text{subject to} \quad z_\alpha a_\alpha + a_\beta^T z_\beta = c_1 \tag{29}$$

$$-\vec{\ell} \leq (z_\alpha, z_\beta) \leq \vec{\ell} \tag{30}$$

where $a = [a_\alpha, a_\beta^T]^T$.

The RGM minimizes the objective function iteratively only in terms of independent variables. The movement of a dependent variable is dictated by the relationship in (29). That is, if $z_\beta(k+1) = z_\beta(k) + \Delta z_\beta$, and $z_\alpha(k+1) = z_\alpha(k) + \Delta z_\alpha$, where $k$ is an iteration index, then

$$\Delta z_\alpha a_\alpha + a_\beta^T \Delta z_\beta = 0$$

or

$$\Delta z_\alpha = -\frac{1}{a_\alpha} a_\beta^T \Delta z_\beta. \tag{31}$$

The movement controlled by (31) guarantees that the new point, $(z_\alpha(k+1), z_\beta(k+1))$, is always on the $(q-1)$-dimensional hyperplane given in (29).

The steepest descent method is now used to find the direction of the independent variable in the RGM. The gradient of each variable determines the direction of movement at each step of the steepest descent method. The gradient of $J(z)$ follows as

$$\nabla_z J(z) = (\nabla_{z_\alpha} J(z), \nabla_{z_\beta} J(z)) = Kz = K(z_\alpha, z_\beta).$$

Since the relationship between the movements of $z_\alpha$ and $z_\beta$ should satisfy (31), the amount of movement, $\nabla_{z_\alpha} J(z)$, for $z_\alpha$ can be decomposed into the amount of movement for $z_\beta$. The gradient with respect to $z_\beta$ (the *reduced gradient* of $z$), instead of $z$, is found to be

$$r^T = \nabla_{z_\beta} J(z_\alpha, z_\beta) - \frac{1}{a_\alpha} \nabla_{z_\alpha} J(z_\alpha, z_\beta) a_\beta \tag{32}$$

where $r = [r_1, r_2, \cdots, r_{q-1}]^T$.

Since the boundary is a box, each variable has a corresponding boundary defined by two sides of the box. In order to allow the possibility that an independent variable is about to violate the boundary constraint, we introduce the *working set* concept. A constraint is defined as *active* if the corresponding variable reaches its constraint boundary which it is about to violate. At each step of the iteration, a subset of boundary constraints that are *active* at the current step is chosen. We call this the working set, denoted by $\mathcal{W}(z_\beta)$.

The vector $z_\beta$ moves in the direction of its gradient unless it violates its boundary constraint. That is, at each step, the direction of $z_\beta$'s movement is

$$\Delta z_{\beta,i} = \begin{cases} -r_i, & i \notin \mathcal{W}(z_\beta) \\ 0, & i \in \mathcal{W}(z_\beta). \end{cases}$$

If an independent variable reaches an edge of its boundary constraint, it becomes part of the working set. This pro-

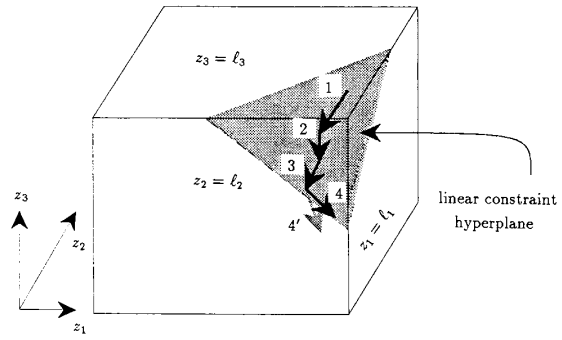



Fig. 2. Boundary constraints and searching.

cess is illustrated in Fig. 2. The numbers in Fig. 2 denote the resultant minimum cost points from the RGM. During iteration, intermediate solutions move as $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ and the boundary of $z_2$ is reached. This occurs, even though the gradient information implies use of path 4′ to minimize the given cost function. The intermediate solution should not choose the path 4′ since it violates the boundary constraint. Instead, by choosing path 4, the solution guarantees $z_2 \leq \ell_2$. In this case, the constraint $z_2 \leq \ell_2$ is called active and the corresponding variable $z_2$ is included in the working set. A more detailed explanation can be found in [9].

Assume that a point is found to be such that

$$r_i = 0 \qquad \forall i \notin \mathcal{W}(z_\beta)$$

but there exists $j \in \mathcal{W}(z_\beta)$ such that either $r_j < 0$ and $j$ have been put into the working set because the corresponding variable violated its lower boundary constraint, or $r_j > 0$ and $j$ have been put into the working set because the corresponding variable violated its upper boundary constraint. Then $j$ is deleted from the working set. Once the amount of movement, $\Delta z_\beta$, of $z_\beta$ is found, $\Delta z_\alpha$ is given by (31).

The RGM generally converges to a relative minimum in most of the cases. There also exists a modified RGM which yields global convergence [9]. The convex simplex method is another variation of RGM which guarantees global convergence [12]. We found little performance variation in our simulations.

We desire the solution by the RGM to be the global minimum within the boundary constraint. This is ensured if the boundary constraint $\mathcal{B}$ is a convex set and the cost function, $J(z)$, is a convex function. Both conditions are present in our problem. We investigate the globalness of the relative minimum obtained with RGM by utilizing the convexity properties of the cost function, $J(z)$, and boundary constraint, $\mathcal{B}$. The following definitions establish the convexity of a function and a set.

*Definition 1 (Convex Set):* A set $\Omega$ in $E^n$ is said to be convex if $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \Omega$ and for $0 \leq \alpha \leq 1$, the point

$$\boldsymbol{x}_3 = \alpha \boldsymbol{x}_1 + (1 - \alpha)\boldsymbol{x}_2$$

is also in $\Omega$. Clearly, since $\mathcal{B}$ is a box, it is convex.

TABLE I
A SUMMARY ATNN ALGORITHM

0. Begin
1. Calculate $\boldsymbol{A}$ and $\boldsymbol{S}$ by using $N$ data as described in (23) and (24).
2. Find $\boldsymbol{K}$ such that $\boldsymbol{K} = \boldsymbol{S}^T(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{S}$.
3. By using (13)–(14), find the linear constraint equation:
$$z^T\boldsymbol{a} = c_1.$$
4. Find the boundary constraint, $\vec{\ell} = 2\hat{z}$, by using
$$\hat{z} = \frac{c_1\boldsymbol{a}}{\boldsymbol{a}^T\boldsymbol{a}}.$$
5. Perform the reduced gradient method and find $z$ which minimizes the cost function:
$$J(z) = \tfrac{1}{2}z^T\boldsymbol{K}z$$
with the constraints $z^T\boldsymbol{a} = c_1$ and $-\vec{\ell} \leq z \leq \vec{\ell}$.
6. Find $\Delta\boldsymbol{W}$ and $\Delta\boldsymbol{v}$ by rearranging $z$ and resulting $\Delta\boldsymbol{W}_{\text{vec}}$.
$$[\Delta\boldsymbol{W}_{\text{vec}}^T; \Delta\boldsymbol{v}^T] = z^T.$$
7. Update $\boldsymbol{W}(N)$ and $\boldsymbol{v}(N)$:
$$\boldsymbol{W}(N+1) = \boldsymbol{W}(N) + \Delta\boldsymbol{W}$$
$$\boldsymbol{v}(N+1) = \boldsymbol{v}(N) + \Delta\boldsymbol{v}.$$
8. Stop.

*Definition 2 (Convex Function):* A function $f$ defined on convex set $\Omega$ is said to be convex if $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \Omega$ and $\forall \alpha \in \mathcal{R}$, $0 \leq \alpha \leq 1$,

$$f(\alpha\boldsymbol{x}_1 + (1 - \alpha)\boldsymbol{x}_2) \leq \alpha f(\boldsymbol{x}_1) + (1 - \alpha) f(\boldsymbol{x}_2).$$

*Lemma 1:* The cost function $J(z)$ is convex. (A proof of Lemma 1 is given in Appendix II).

Since $J(z)$ is a convex function defined on the convex set $\Omega$, any achieved relative minimum of $J(z)$ is the global minimum.

A summary of the adaptively trained ATNN is given in Tables I and II.

## VII. Susceptibility to Drift

A recursive algorithm for adaptive training must possess two features: it should adapt to new data and it should not drift. Drift is an undesired feature, wherein the weights of the layered perceptron change even when the new data do not change the input/output relationship of the existing neural network.

Assume that the trained network with $\{\boldsymbol{W}(N), \boldsymbol{v}(N)\}$ has the following functional relationship between the input, $x$, and the output, $y$:

$$y = f(\boldsymbol{W}(N), \boldsymbol{v}(N); x) \tag{33}$$

and

$$D = \{(x(i), d(i)); 1 \leq i \leq N\}.$$

If we use additional data, $(x(N+1), d(N+1))$, that are not in $D$ but satisfy (33), both $\Delta\boldsymbol{W}$ and $\Delta\boldsymbol{v}$ should be equal to $\boldsymbol{0}$. Otherwise, the proposed adaptive procedure is said to drift. We will now show that our proposed algorithm does not drift.

From (11) and (13),

$$y = d(N+1) = f(\boldsymbol{W}(N), \boldsymbol{v}(N); x(N+1))$$

TABLE II
A SUMMARY OF THE REDUCED GRADIENT METHOD

0. Set initial feasible solution $z = \hat{z}$.
1. Set $z_\alpha = z_1$ and $z_\beta = [z_2, z_3, \cdots, z_q]^T$. Initialize $\mathcal{W}(z_\beta) = \{\phi\}$, where $\{\phi\}$ denotes the empty set.
2. Calculate

$$r^T = \nabla_{z_\beta} J(z_\alpha, z_\beta) - \frac{1}{a_\alpha} \nabla_{z_\alpha} J(z_\alpha, z_\beta) a_\beta.$$

3. Find the direction of the movement of $z_\beta$:

$$\Delta z_{\beta,i} = \begin{cases} -r_i, & i \notin \mathcal{W}(z_\beta) \\ 0, & i \in \mathcal{W}(z_\beta). \end{cases}$$

4. Reconstruct the working set $\mathcal{W}(z_\beta)$ if necessary.
5. We stop if the point is an acceptable solution. This occurs when $|\Delta z| \leq \epsilon$, where $\epsilon$ is a convergence measure, or $\forall i \in \mathcal{W}(z_\beta)$. Otherwise, find

$$\Delta z_\alpha = -\frac{1}{a_\alpha} a_\beta^T \Delta z_\beta.$$

6. Find $\gamma_1$, $\gamma_2$, and $\gamma_3$ such that

$$\max \{\gamma_1: -l_\alpha \leq z_\alpha + \gamma_1 \Delta z_\alpha \leq l_\alpha;\ \gamma_1 \geq 0\}$$

$$\max \{\gamma_2: -\vec{\ell}_\beta \leq z_\beta + \gamma_2 \Delta z_\beta \leq \vec{\ell}_\beta;\ \gamma_2 \geq 0\}$$

$$\min \{\gamma_3: J(z + \gamma_3 z);\ 0 \leq \gamma_3 \leq \gamma_1,\ 0 \leq \gamma_3 \leq \gamma_2\}.$$

7. Calculate $z = z + \gamma_3 z$.
8. If $\gamma_3 < \gamma_1$, go to 2.
   Otherwise, declare the dependent variable to be independent and declare one of the independent variables, which is positively inside of the nonlinearity constraint, to be dependent. Update $a_\alpha$ and $a_\beta$. Go to 2.

and we have

$$c_1 = 0.$$

Consequently, $z^T a = 0$ becomes the line constraint for this problem. From (26), the closest point which satisfies this line constraint is

$$\hat{z} = \frac{c_1 a}{a^T a} = \mathbf{0}.$$

Since $\hat{z} = \mathbf{0}$ gives

$$J = \tfrac{1}{2} z^T K z = 0$$

the initial point to the RGM, $\hat{z} = \mathbf{0}$, is the solution. This implies that

$$\Delta W = \mathbf{0} \quad \text{and} \quad \Delta v = \mathbf{0}.$$

Therefore, we conclude that there is no drifting in the ATNN. Note, alternatively, that the drift problem can also be addressed by triggering the ATNN only when the output error exceeds some specified threshold.

## VIII. Choosing Weights on the $E_i$'s

The weighting coefficient, $\gamma_i$, on each of the error terms of (3) can be used to weight the error function as follows:

$$\hat{E}(N) = \tfrac{1}{2} \sum_{i=1}^{N} \gamma_i (d(i) - y(i))^2 = \sum_{i=1}^{N} \hat{E}_i$$

where $\hat{E}(N)$ is a weighted version of $E(N)$ and

$$\hat{E}_i = \tfrac{1}{2} \gamma_i (d(i) - y(i))^2.$$

By introducing this weighting coefficient, $\gamma_i = \mathfrak{F}(i)$, we can control the comparative importance of the previous data.

The weighting coefficients can be incorporated into the ATNN as follows. Since

$$\hat{E}_i = \tfrac{1}{2} \gamma_i (d(i) - y(i))^2 = \gamma_i E_i$$

we can rewrite (22) with the WCGF:

$$\Delta \hat{E} = \Gamma A S z = \hat{A} S z$$

where $\hat{A} = \Gamma A$ and

$$\Gamma = \text{diag}\,[\gamma_1, \gamma_2, \cdots, \gamma_N].$$

Therefore, we can consider the weighting coefficients in the ATNN by simply changing $A$ to $\hat{A}$.

## IX. Experiments and Results

Since Lapedes and Farber [14] suggested their use for the system identification, the layered perceptron has been successfully applied to many other system identification problems [14]–[23]. In the following example ATNN applications, the layered perceptron is used for system identification.

First, an examplar problem that has single input and single output is considered. It nicely illustrates the proposed algorithm graphically. Then, the ATNN is applied to the electric load forecasting problem in order to illustrate its application to highly nonstationary training data.

### A. Examplar Problem

We are given a set of one-dimensional input and one-dimensional output data such that

$$\{(x(1), d(1)), (x(2), d(2)), \cdots, (x(100), d(100))\}.$$

where $x(n) = 0.01(n - 1)$. A 3-layered perceptron with ten hidden neurons was trained using error back-propagation to obtain $W(100)$ and $v(100)$, the weights for the given 100 data points.

In order to illustrate the ATNN, assume that a new data point $(x(101), d(101))$ is to be used to update the network weights and that the new data are

$$x(101) = x(51) = 0.5$$

and

$$d(101) = 1.1 \times d(51) = 0.45 + 0.045 = 0.495.$$

Thus, the new data are inconsistent with some of previous training data.

*Results:* We applied our algorithm to the above prob-

TABLE III
THE RESULTANT CHANGE IN WEIGHTS FROM USING THE ATNN

| | Old | Δ | New | | Old | Δ | New |
|---|---|---|---|---|---|---|---|
| $W_{0,2}$ | −1.6941 | 0.0274 | −1.6667 | $W_{1,2}$ | −2.3778 | −0.0006 | −2.3785 |
| $W_{0,3}$ | −1.2054 | 0.0514 | −1.1539 | $W_{1,3}$ | −0.5910 | −0.0549 | −0.6459 |
| $W_{0,4}$ | −1.1859 | 0.2209 | −0.9649 | $W_{1,4}$ | −7.5534 | −0.2781 | −7.2752 |
| $W_{0,5}$ | 15.3013 | −0.4345 | 14.8667 | $W_{1,5}$ | −22.4426 | 0.1881 | −22.2545 |
| $W_{0,6}$ | −1.0096 | 0.0190 | −0.9906 | $W_{1,6}$ | −0.9404 | −0.0181 | −0.9586 |
| $W_{0,7}$ | −1.1323 | 0.0375 | −1.0947 | $W_{1,7}$ | −0.4236 | −0.0484 | −0.4720 |
| $W_{0,8}$ | −1.0451 | 0.0260 | −1.0191 | $W_{1,8}$ | −0.4856 | −0.0337 | −0.5193 |
| $W_{0,9}$ | −2.0645 | 0.1131 | −1.9514 | $W_{1,9}$ | −3.9084 | 0.0350 | −3.8733 |
| $W_{0,10}$ | 2.1602 | 0.0125 | 2.1727 | $W_{1,10}$ | −14.4980 | −0.2369 | −14.7349 |
| $W_{1,11}$ | −0.9274 | 0.0330 | −0.8944 | $W_{1,11}$ | −0.5305 | −0.0427 | −0.5732 |
| $v_0$ | −1.4489 | −0.3237 | −1.7726 | $v_1$ | −0.6528 | 0.6081 | −0.0446 |
| $v_2$ | −0.1127 | −0.1584 | −0.2712 | $v_3$ | −0.4850 | −0.1392 | −0.6242 |
| $v_4$ | −4.1520 | −0.0143 | −4.1376 | $v_5$ | 2.0000 | 0.3647 | 2.3647 |
| $v_6$ | −0.0708 | −0.1916 | −0.2625 | $v_7$ | −0.3882 | −0.1280 | −0.5163 |
| $v_8$ | −0.2301 | −0.1413 | −0.3715 | $v_9$ | −1.1492 | −0.0908 | −1.2400 |
| $v_{10}$ | 1.9850 | 0.4237 | 2.4087 | $v_{11}$ | −0.2795 | −0.1582 | −0.4377 |

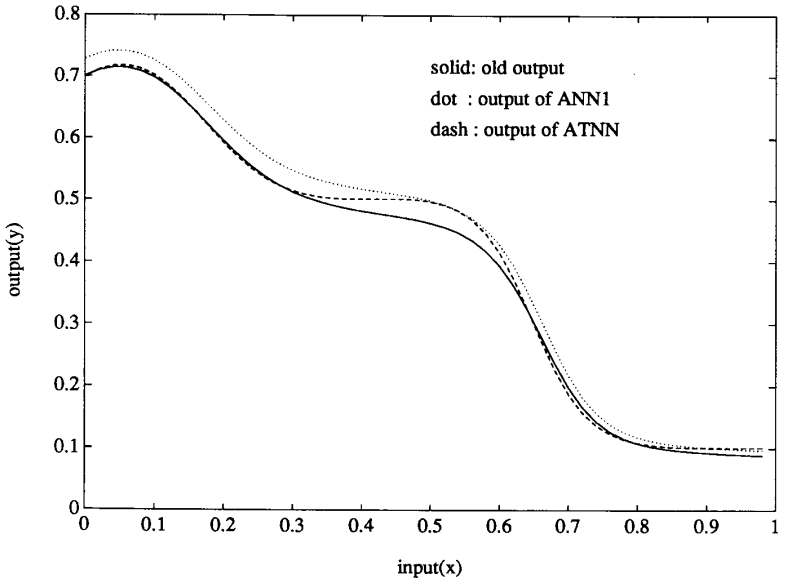



Fig. 3. A performance comparison of the ATNN (dashed line) to (a) previously trained layered perceptron (solid line) and (b) a layered perceptron retained with new data (dotted line). The performance of a layered perceptron retained with both old and new data is graphically indistinguishable from (a).

lem and found a set of optimal weights, $\boldsymbol{W}(101)$ and $\boldsymbol{v}(101)$, as shown in Table III. In order to show the effects of the new weights on the previous data, we test the layered perceptron with the new weights. Fig. 3 shows the changes in outputs for the previous data. Note that the layered perceptron responds to the desired output of the new data despite inconsistency between the old and the new data. (The representation obtained from the neural net trained with the old data is graphically indistinguishable from the training data.)

A comparison is also made between the ATNN and two other training approaches. In the first alternative method, a layered perceptron is retrained with only the new data point using the old network as a point of initialization. This guarantees that the network, after retraining, produces a desirable output for the new data. As illustrated in Fig. 3, the retrained network, however, does not maintain proper performance for the old data. Yet another layered perceptron was trained with both the old and new data. The layered perceptron trained with the old data was used for the initialization. The retrained network with this technique, however, has a performance that is graphically indistinguishable from that of the performance of the layered perceptron trained only on the old data.

### B. Application to Load Forecasting

Electric load forecasting is used by electric utility companies for prediction of electric power consumption based on pertinent data such as past consumption and weather information.

Electric system load forecasting with lead times from hours to several days can assist a power system operator to efficiently schedule spinning reserve allocation. In addition, the ability to forecast electric system load can also provide valuable information for possible energy interchange with other utilities. Accurate information for future loads is also useful in detecting many vulnerable situations in advance if applied to system security assessment problem. Because of the importance of load forecasting, forecasts should be accurate and sufficiently timely to assist the system operator.

*1) Approaches:* Previous approaches to load forecasting in general fall into three categories. First, the load pattern is treated as a time series signal and future loads are predicted by using various time series analysis techniques [24]–[26]. The second approach assumes that the load pattern is highly correlated with such weather variables as temperature and wind speed and finds a functional relationship between weather variables and the system load. The future load can then be forecast by inserting the predicted weather information into the predetermined functional relationship [27]–[30]. The last approach combines the time series approach and the regression approach by using artificial neural networks. More details on this application and corresponding results can be found in [22], where forecasting was performed by training a layered perceptron with conventional error back-propagation. A comparative study on different problems, including the load forecasting problem, contrasting the performance of a layered perceptron and the classification and regression trees (CART) has been recently reported by Atlas *et al.* [23].

*2) Experiments:* Hourly temperature and load data for the Seattle/Tacoma area from November 1, 1989, to February 9, 1990, were collected and provided for us by the Puget Sound Power and Light Company. Our focus is on hourly load forecasting with lead time of 48 hours for a normal weekday (i.e., no holidays or weekends). All data except the last nine days (February 1–9) are used for training the layered perceptron, and the data of the last nine days are used for testing the trained layered perceptron. As a topology for the layered perceptron, a good choice of input variables for the output, $L_k$, was found to be

$$(k,\ T_{k-168},\ T_{k-50},\ T_{k-49},\ T_{k-48},$$
$$L_{k-168},\ L_{k-50},\ L_{k-49},\ L_{k-48},\ T_k)$$

where $k$ is the hour of the day and $T_i$ and $L_j$ represents the temperature and load at hour $i$ and $j$ respectively. The layered perceptron used in this experiment has 11 inputs (including a bias term), five hidden neurons, and one output neuron.

TABLE IV
A PERFORMANCE COMPARISON OF THE ATNN TO REGULAR LAYERED PERCEPTRON BY THE AVERAGE OF ABSOLUTE ERROR (%) IN THE ELECTRIC LOAD FORECASTING PROBLEM

| | 2/1 | 2/2 | 2/7 | 2/8 | 2/9 |
|---|---|---|---|---|---|
| ANN | 6.14 | 6.53 | 5.28 | 2.92 | 2.36 |
| ATNN | 2.18 | 2.26 | 4.65 | 2.36 | 2.79 |

First, a standard error back-propagation algorithm is used for training the layered perceptron. The ATNN algorithm is then applied to the trained layered perceptron to improve the performance. The ATNN adapts the trained layered perceptron by sequentially using the latest 24 data points while the regular layered perceptron simply augments the training data set with 24 new entries.

*3) Results:* The actual temperatures are used in the testing stage to find the effects of the ATNN algorithm on the forecast loads since forecast temperatures deviate from actual temperature by as much as 14% (February 1, 1990). Use of forecast temperatures therefore distracts the ATNN.

Daily average errors are calculated as

$$\text{percent error} = \frac{1}{24}\sum_{k=1}^{24}\frac{|L_k - L_k'|}{L_k} \times 100\%$$

where $L_k$ and $L_k'$ denote actual and forecast loads at hour $k$.

As shown in Table IV, the comparative results imply that the ATNN gives improved results except for day 5. The improvement is especially significant on day 2, when the weather changed abruptly. Fig. 4 depicts a comparative example of five days of forecasts.

These results were generated using a value of $k = 2$. As with the step size and momentum terms in error back-propagation, the choice of a good value of $k$ remains somewhat of an art.

## X. CONCLUSION

In this paper, an adaptive training algorithm for a layered perceptron has been proposed. The algorithm uses a previously trained layered perceptron to adapt the weights for the new data. Conventional learning algorithms, such as back-propagation, can give an averaged result for inconsistent data, which can occur often in a nonstationary environment.

This adaptively trained neural network (ATNN) utilizes nonlinear programming methods to adapt to temporally nonstationary training data.

We have demonstrated the use of the ATNN by applying it to the nonstationary problem of electric load forecasting, where the seasonal or annual load change can be severe. The ATNN has the additional flexibility of allowing importance weighting on training data.

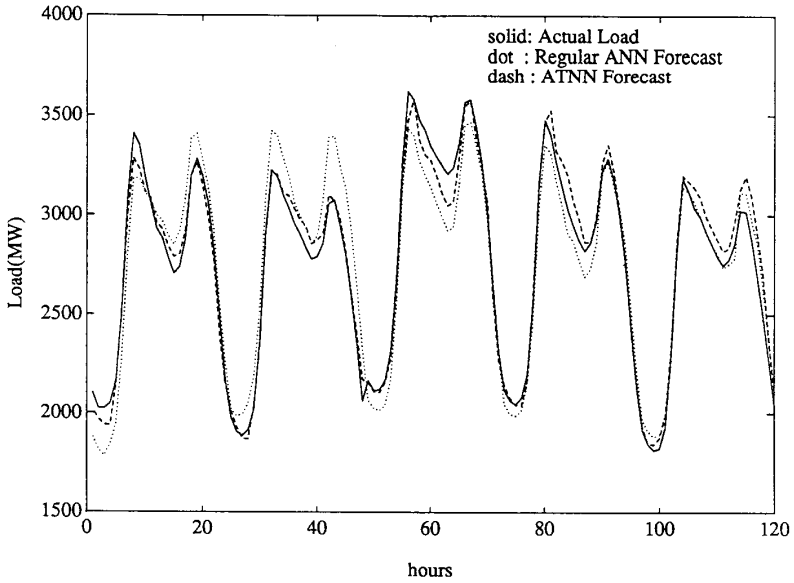



Fig. 4. A performance comparison of the ATNN (dashed line) to conventional layered perceptron (dotted line) when applied to electric load forecasting problem. Solid line represents actual load.

## Appendix I
## Sensitivity Matrix for the General Case

Consider a layered perceptron which has $n$ hidden layers and each hidden layer has $N_i$ hidden neurons. Assume that $w_{i,j}^{(m)}$ is the weight of connection between neuron $i$ in the $m$th hidden layer from the output layer and the neuron $j$ in the $(m-1)$st hidden layer from the output layer. Then, $\boldsymbol{u}^{(i)} = [u_1^{(i)}, u_2^{(i)}, \cdots, u_{N_i}^{(i)}]^T$, where $N_i$ represents the number of neurons in the $i$th hidden layer from the output layer and $u_j^{(i)}$ denotes the activation of the neuron $j$ in that hidden layer. Then the sensitivity of the $\ell$th output, $y_\ell$, by the change of $w_{i,j}^{(m)}$ is given by the following.

*1) For $m = 1$:*

$$\frac{\partial y_\ell}{\partial w_{i,\ell}^{(m)}} = \frac{\partial f\left[\sum_{j=1}^{N_1} w_{j,\ell}^{(1)}\right]}{\partial w_{i,\ell}^{(1)}} = y_\ell(1 - y_\ell)u_i^{(1)}.$$

*2) $m = 2$:*

$$\frac{\partial y_\ell}{\partial w_{i,j}^{(m)}} = \left(\frac{\partial y_\ell}{\partial w_j^{(1)}}\right)\left(\frac{\partial u_j^{(1)}}{\partial u_i^{(2)}}\right).$$

*3) For $m \geq 3$:*

$$\frac{\partial y_\ell}{\partial w_{i,j}^{(m)}} = \nabla_{\boldsymbol{u}^{(1)}} y_\ell \left\{\prod_{k=1}^{m-3}\left(\frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}}\right)\right\}\left(\frac{\partial \boldsymbol{u}^{(m-2)}}{\partial u_j^{(m-1)}}\right)\left(\frac{\partial u_j^{(m-1)}}{\partial u_i^{(m)}}\right) \quad (34)$$

where

$$u_p^{(k)} = f\left[\sum_{q=1}^{N_{(k+1)}} w_{q,p}^{(k+1)} u_q^{(k+1)}\right]$$

$$\frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} = \frac{\partial [u_1^{(k)}, u_2^{(k)}, \cdots, u_{N_k}^{(k)}]^T}{\partial \boldsymbol{u}^{(k+1)}}$$

$$= \begin{bmatrix} \nabla_{\boldsymbol{u}^{(k+1)}} u_1^{(k)} \\ \vdots \\ \nabla_{\boldsymbol{u}^{(k+1)}} u_{N_k}^{(k)} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial u_1^{(k)}}{\partial u_1^{(k+1)}} & \cdots & \frac{\partial u_1^{(k)}}{\partial u_{N(k+1)}^{(k+1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_{N_k}^{(k)}}{\partial u_1^{(k+1)}} & \cdots & \frac{\partial u_{N_k}^{(k)}}{\partial u_{N(k+1)}^{(k+1)}} \end{bmatrix}$$

$$= \begin{bmatrix} \delta_1^{(k)} w_{1,1}^{(k+1)} & \cdots & \delta_1^{(k)} w_{N(k+1),1}^{(k+1)} \\ \vdots & \ddots & \vdots \\ \delta_{N_k}^{(k)} w_{1,N_k}^{(k+1)} & \cdots & \delta_{N_k}^{(k)} w_{N(k+1),N_k}^{(k+1)} \end{bmatrix}$$

and

$$\delta_i = \left.\frac{\partial f[x]}{\partial x}\right|_{x = u_i^{(k)}} = u_i^{(k)}(1 - u_i^{(k)}).$$

*Proof:* For the cases where $m > 3$, define

$$\frac{\partial u_j^{(k)}}{\partial u_i^{(k+1)}} \overset{\text{def}}{=} \gamma(i(k+1), j(k)).$$

Since

$$\frac{\partial \boldsymbol{u}^{(k-1)}}{\partial \boldsymbol{u}^{(k+1)}} = \left(\frac{\partial \boldsymbol{u}^{(k-1)}}{\partial \boldsymbol{u}^{(k)}}\right)\left(\frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}}\right)$$

the sensitivity of $u_n^{(k-1)}$ to the change of $u_i^{(k+1)}$ is given by

$$\frac{\partial u_n^{(k-1)}}{\partial u_i^{(k+1)}} = \left(\frac{\partial u_n^{(k-1)}}{\partial \boldsymbol{u}^{(k)}}\right)\left(\frac{\partial \boldsymbol{u}^{(k)}}{\partial u_i^{(k+1)}}\right)$$

$$= \nabla_{\boldsymbol{u}^{(k)}} u_n^{(k-1)} \begin{bmatrix} \dfrac{\partial u_1^{(k)}}{\partial u_i^{(k+1)}} \\ \vdots \\ \dfrac{\partial u_{N_m}^{(k)}}{\partial u_i^{(k+1)}} \end{bmatrix}$$

$$= \nabla_{\boldsymbol{u}^{(k)}} u_n^{(k-1)} \begin{bmatrix} \gamma(i(k+1), 1(k)) \\ \vdots \\ \gamma(i(k+1), N_k(k)) \end{bmatrix}.$$

---

$$\frac{\partial \boldsymbol{u}^{(k-1)}}{\partial \boldsymbol{u}^{(k+1)}} = \left[\frac{\partial \boldsymbol{u}^{(k-1)}}{\partial u_1^{(k+1)}}, \cdots, \frac{\partial \boldsymbol{u}^{(k-1)}}{\partial u_{N(k-1)}^{(k+1)}}\right]$$

$$= \frac{\partial [u_1^{(k+1)} \cdots, u_{N(k-1)}^{(k-1)}]^T}{\partial \boldsymbol{u}^{(k)}}$$

$$\cdot \begin{bmatrix} \gamma(1(k+1), 1(k)) & \cdots & \gamma(N_{(k+1)}(k+1), 1(k)) \\ \vdots & \ddots & \vdots \\ \gamma(1(k+1), N_k(k)) & \cdots & \gamma(N_{(k+1)}(k+1), N_k(k)) \end{bmatrix}$$

$$= \frac{\partial \boldsymbol{u}^{(k-1)}}{\partial \boldsymbol{u}^{(k)}} \begin{bmatrix} \dfrac{\partial u_1^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} \\ \vdots \\ \dfrac{\partial u_{N_k}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} \end{bmatrix}$$

$$= \left(\frac{\partial \boldsymbol{u}^{(k-1)}}{\partial \boldsymbol{u}^{(k)}}\right)\left(\frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}}\right).$$

Thus, the sensitivities of activation in the $(k-1)$st hidden layer by a change in $u_i^{(k+1)}$ are

$$\frac{\partial \boldsymbol{u}^{(k-1)}}{\partial u_i^{(k+1)}} = \frac{\partial}{\partial u_i^{(k+1)}} [u_1^{(k-1)}, \cdots, u_{N(k-1)}^{(k-1)}]^T$$

$$= \left[\frac{\partial u_1^{(k-1)}}{\partial u_i^{(k+1)}}, \cdots, \frac{\partial u_{N(k-1)}^{(k-1)}}{\partial u_i^{(k+1)}}\right]^T$$

$$= \begin{bmatrix} \nabla_{\boldsymbol{u}^{(k)}} u_1^{(k-1)} \begin{bmatrix} \gamma(i(k+1), 1(k)) \\ \vdots \\ \gamma(i(k+1), N_k(k)) \end{bmatrix} \\ \vdots \\ \nabla_{\boldsymbol{u}^{(k)}} u_{N_{(k-1)}}^{(k-1)} \begin{bmatrix} \gamma(i(k+1), 1(k)) \\ \vdots \\ \gamma(i(k+1), N_k(k)) \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \nabla_{\boldsymbol{u}^{(k)}} u_1^{(k-1)} \\ \vdots \\ \nabla_{\boldsymbol{u}^{(k)}} u_{N_{(k-1)}}^{(k-1)} \end{bmatrix} \begin{bmatrix} \gamma(i(k+1), 1(k)) \\ \vdots \\ \gamma(i(k+1), N_k(k)) \end{bmatrix}$$

$$= \frac{\partial [u_1^{(k-1)}, \cdots, u_{N(k-1)}^{(k-1)}]^T}{\partial \boldsymbol{u}^{(k)}}$$

$$\cdot \begin{bmatrix} \gamma(i(k+1), 1(k)) \\ \vdots \\ \gamma(i(k+1), N_k(k)) \end{bmatrix}$$

As a result, the sensitivities of activations for the neurons in the $(k-1)$st hidden layer by the change of activations for the neurons in the $(k+1)$st hidden layer are

Therefore, the sensitivities of activation for the neurons from one layer by the changes of activations for the neurons to the other layer can be calculated in a cascaded fashion. That is,

$$\frac{\partial \boldsymbol{u}^{(0)}}{\partial \boldsymbol{u}^{(m)}} = \prod_{k=0}^{m-1} \frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} = \frac{\partial \boldsymbol{u}^{(0)}}{\partial \boldsymbol{u}^{(1)}} \left\{ \prod_{k=1}^{m-3} \left( \frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} \right) \right\} \cdot \left( \frac{\partial \boldsymbol{u}^{(m-2)}}{\partial \boldsymbol{u}^{(m-1)}} \right) \left( \frac{\partial \boldsymbol{u}^{(m-1)}}{\partial \boldsymbol{u}^{(m)}} \right) \tag{35}$$

where $\boldsymbol{u}^{(0)}$ are the activations of neurons in output layer.

Since the change of $w_{i,j}^{(m)}$ affects only $u_j^{(m-1)}$ by an amount $\partial u_j^{(m-1)}/\partial w_{i,j}^{(m)}$, the change of $u_j^{(m-1)}$ affects all the activations of neurons in the $(m-1)$st hidden layer by the amount of $\partial \boldsymbol{u}^{(m-2)}/\partial u_j^{(m-1)}$, the change of $\boldsymbol{u}^{(m-2)}$ affects the neurons in the first hidden layer by $\partial \boldsymbol{u}^{(1)}/\partial \boldsymbol{u}^{(m-2)}$, and, finally, the change of $\boldsymbol{u}^{(1)}$ affects the output neuron $y_\ell$ by $\partial y_\ell/\partial \boldsymbol{u}^{(1)}$. Therefore,

$$\frac{\partial y_\ell}{\partial w_{i,j}^{(m)}} = \frac{\partial y_\ell}{\partial \boldsymbol{u}^{(1)}} \left\{ \prod_{k=1}^{m-3} \left( \frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} \right) \right\} \left( \frac{\partial \boldsymbol{u}^{(m-2)}}{\partial u_j^{(m-1)}} \right) \left( \frac{\partial u_j^{(m-1)}}{\partial u_i^{(m)}} \right) = \nabla_{\boldsymbol{u}^{(1)}} y_\ell \left\{ \prod_{k=1}^{m-3} \left( \frac{\partial \boldsymbol{u}^{(k)}}{\partial \boldsymbol{u}^{(k+1)}} \right) \right\} \left( \frac{\partial \boldsymbol{u}^{(m-2)}}{\partial u_j^{(m-1)}} \right) \left( \frac{\partial u_j^{(m-1)}}{\partial u_i^{(m)}} \right).$$

This concludes the proof of (34). Q.E.D.

By using the sensitivity of each weight to the output, the error resulting from the perturbated weights can be found as in (20):

$$\Delta E_i = \sum_{j,k,m} \left( \frac{\partial E_i}{\partial w_{j,k}^{(m)}} \right) \Delta w_{j,k}^{(m)}.$$

The sensitivity matrix $\boldsymbol{S}$ for the general case can also be found by the same method applied for the single hidden layer case by using (22) to (24).

## Appendix II
## Proof of Lemma 1 (Convexity of the Cost Function)

For our case,

$$J(z) = \tfrac{1}{2} z^T \boldsymbol{K} z.$$

Thus

$$\begin{aligned} J(\alpha z_1 + (1-\alpha) z_2) &= (\alpha z_1 + (1-\alpha) z_2)^T \boldsymbol{K} (\alpha z_1 + (1-\alpha) z_2) \\ &= \alpha z_1^T \boldsymbol{K} \alpha z_1 + \alpha z_1^T \boldsymbol{K} (1-\alpha) z_2 \\ &\quad + (1-\alpha) z_2^T \boldsymbol{K} \alpha z_1 + (1-\alpha)^2 z_2^T \boldsymbol{K} z_2 \\ &= \alpha^2 z_1^T \boldsymbol{K} z_1 + \alpha(1-\alpha)\left[ z_1^T \boldsymbol{K} z_2 + z_2^T \boldsymbol{K} z_1 \right] \\ &\quad + (1-\alpha)^2 z_2^T \boldsymbol{K} z_2. \end{aligned} \tag{36}$$

Let

$$\mathfrak{A} = \alpha J(z_1) + (1-\alpha) J(z_2)$$

then

$$\mathfrak{A} = \alpha z_1^T \boldsymbol{K} z_1 + (1-\alpha) z_2^T \boldsymbol{K} z_2.$$

Also,

$$\begin{aligned} \mathfrak{A} &= \alpha \mathfrak{A} + (1-\alpha)\, \mathfrak{A} \\ &= \alpha^2 z_1^T \boldsymbol{K} z_1 + \alpha(1-\alpha) z_2^T \boldsymbol{K} z_2 \\ &\quad + (1-\alpha)\alpha z_1^T \boldsymbol{K} z_1 + (1-\alpha)^2 z_2^T \boldsymbol{K} z_2. \end{aligned} \tag{37}$$

By combining (36) and (37),

$$\begin{aligned} \mathfrak{A} - J(\alpha z_1 + (1-\alpha) z_2) &= \alpha(1-\alpha)\left[ z_1^T \boldsymbol{K} z_1 + z_2^T \boldsymbol{K} z_2 - z_1^T \boldsymbol{K} z_2 - z_2^T \boldsymbol{K} z_1 \right] \\ &= \alpha(1-\alpha)(z_1 - z_2)^T \boldsymbol{K} (z_1 - z_2) \geq 0. \end{aligned}$$

Therefore, $\alpha J(z_1) + (1-\alpha) J(z_2) \geq J(\alpha z_1 + (1-\alpha) z_2)$ and consequently $J(z)$ is a convex function by Definition 2. Q.E.D.

## Acknowledgment

The authors thank M. L. Bruce of the Puget Sound Power and Light Company for providing the data for the electric load forecasting problem and Dr. S. Oh, M. Damborg, J.-N. Hwang, and L. Atlas for their valuable comments. They also wish to thank Dr. H. Rauch and the reviewers for their helpful suggestions in preparing this manuscript.

## References

[1] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. Rumelhart *et al.*, Eds. Cambridge, MA: MIT Press, 1986.

[2] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. New York: Academic Press, 1970.

[3] A. Gelb, *Applied Optimal Estimation*. Cambridge, MA: M.I.T. Press, 1974.

[4] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[5] K. S. P. Kumar and R. Shridar, "On the identification of control systems by the quasi-linearization method," *IEEE Trans. Automat. Contr.*, vol. AC-9, pp. 151-154, 1964.

[6] A. P. Sage and B. R. Eisenberg, "Experiments in nonlinear and nonstationary system identification via quasilinearization and differential approximation," in *Proc. Joint Automat. Contr. Conf.*, 1965, pp. 522-530.

[7] F. R. Gantmacher, *The Theory of Matrices*, vol. 1. New York: Chelsea Publishing, 1960.

[8] J. N. Franklin, *Matrix Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1968.

[9] D. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984, pp. 295-321.

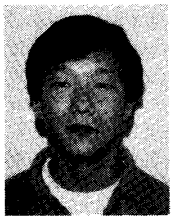
[10] A. V. Fiacco and G. P. McCormick, *Nonlinearing Programming: Sequential Unconstrained Minimization Technique*. New York: Wiley, 1968.

[11] G. P. McCormick, "Optimality criteria in nonlinear programming," in *SIAM-AMS Proc.*, IX, 1976, pp. 27-38.

[12] W. I. Zangwill, *Nonlinear Programming: Unified Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1969.

[13] D. Graupe, *Time Series Analysis, Identification and Adaptive Filtering*. Malabar, FL: Robert E. Kreiger, 1984.

[14] D. Lapedes and R. Farber, "Nonlinear signal processing using neural networks; prediction and system modeling," Los Alamos National Laboratory, Los Alamos, NM, TR LA-UR-87-2662, 1987.
[15] J. Moody and D. Darken, "Learning with localized receptive fields," in *Proc. 1988 Connectionist Summer School*, 1988.
[16] M. Tenrio and W. Lee, "Self-organizing network for optimum supervised learning," *IEEE Trans. Neural Networks*, vol. 1, Mar. 1990.
[17] M. Aggoune, M. El-Sharkawi, D. Park, M. Damborg, and R. Marks II, "Preliminary results on using artificial neural networks for security assessment," in *Proc. 1989 Power Ind. Comp. Appl. Conf.* (Seattle, WA), May 1989. (to appear in *IEEE Trans. Power Systs.*)
[18] M. El-Sharkawi, R. Marks II, M. Aggoune, D. Park, M. Damborg, and L. Atlas, "Dynamic security assessment of power systems using back error propagation artificial neural networks," in *Proc. 2nd Annual Symp. Expert Systems Appl. to Power Systems* (Seattle, WA), July 1989.
[19] C. Lam *et al.*, "Determination of particle size distribution using a neural network trained with backscatter measurement," in *Proc. 1990 AP-S Int. Symp. & URSI Radio Sci. Meeting* (Dallas, TX), May 1990.
[20] A. Ishimaru *et al.*, "Optical sensing of particle size distribution by using artificial neural networks technique," *Opt. Lett.*, vol. 15, pp. 1221-1223, 1990.
[21] L. Atlas *et al.*, "A performance comparison of trained multilayer perceptrons and trained classification trees," in *Proc. IEEE Conf. Syst., Man, Cybern.* (Boston, MA), Nov. 1989.
[22] D. Park, M. El-Sharkawi, R. Marks II, and L. Atlas, "Electric load forecasting using an artificial neural network," *IEEE Trans. Power Syst.*, to be published.
[23] L. Atlas *et al.*, "A performance comparison of trained multi-layer perceptrons and trained classification trees," *Proc. IEEE*, vol. 78, pp. 1614-1619, 1990.
[24] G. E. Box and G. M. Jenkins, *Time Series Analysis—Forecasting and Control*. San Francisco: Holden-Day, 1976.
[25] S. Vemuri, D. Hill, and R. Balasubramanian, "Load forecasting using stochastic models," in *Proc. 8th PICA Conf.* (Minneapolis, MN), 1973, pp. 31-37.
[26] W. Christiaanse, "Short-term load forecasting using general exponential smoothing," *IEEE Trans. Power App. Syst.*, vol. PAS-90, pp. 900-910, Apr. 1971.
[27] P. Gupta and Y. Yamada, "Adaptive short-term forecasting of hourly loads using weather information," *IEEE Trans. Power App. Syst.*, vol. PAS-91, pp. 2085-2094, 1972.
[28] C. Asbury, "Weather load model for electric demand energy forecasting," *IEEE Trans. Power App. Syst.*, vol. PAS-94, pp. 1111-1116, 1975.
[29] S. Rahman and R. Bhatnagar, "An expert system based algorithm for short load forecast," *IEEE Trans. Power Syst.*, vol. 3, pp. 392-399, May 1988.
[30] K. Jabbour, J. Riveros, D. Landbergen, and W. Meyer, "ALFA: Automated load forecasting assistant," *IEEE Trans. Power Syst.*, vol. 3, pp. 908-914, Aug. 1988.



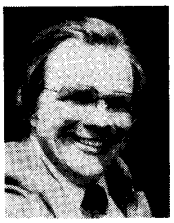
**Dong Chul Park** (S'80-M'90) received the B.S. degree in 1980 from Sogang University, the M.S. degree in 1982 from the Korea Advanced Institute of Science and Technology, Seoul, Korea, and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in June 1990.

He joined the faculty of the Department of Electrical and Computer Engineering at Florida International University, the State University of Florida at Miami, in August 1990. Dr. Park was a committee member for the First International Forum on Applications of Neural Networks to Power Systems (Seattle, 1991). Since 1988 he has published 11 archival journal and 15 proceedings papers in the areas of neural networks, optical computing, signal processing, and electric load forecasting.

**Mohamed A. El-Sharkawi** (S'76-M'80-SM'83) was born in Cairo, Egypt, in 1948. He received the B.Sc. degree in electrical engineering in 1971 from the Cairo High Institute of Technology, Egypt. His M.A.Sc and Ph.D. degrees in electrical engineering were received from the University of British Columbia in 1977 and 1980 respectively.

Since 1980, he has been with the Department of Electrical Engineering at the University of Washington, where he is currently a Professor. He is a pioneer in the area of NN applications to power systems. He is the Chairman of the IEEE Power Engineering Society task force on applications of neural networks to power systems. He has organized and chaired several special sessions and panel discussions on the subject at various IEEE conferences. He was the organizer and chairman of the first conference to be held in Seattle, in 1990, dedicated to the applications of neural networks to power systems.

Dr. El-Sharkawi has published numerous papers in the areas of neural network applications to power systems, power system dynamics, power electronics, electric drives, and high-performance tracking and control.



**Robert J. Marks II** (S'76-M'77-SM'83) holds the title of Professor in the Department of Electrical Engineering at the University of Washington, Seattle. He was awarded the Outstanding Branch Councillor award in 1982 by the IEEE and, in 1984, was presented with an IEEE Centennial Medal. He was also the Chair of the IEEE Neural Networks Committee and was the cofounder and first Chair of the IEEE Circuits & Systems Society Technical Committee on Neural Systems & Applications. Dr. Marks was also elected the first President of the IEEE Council on Neural Networks. He is a Fellow of the Optical Society of America. He also was a cofounder and first President of the Puget Sound Section of the Optical Society of America and was elected that organization's first honorary member. He is cofounder and current President of the Multidimensional Systems Corporation in Bellevue, WA.

Dr. Marks is the topical editor for Optical Signal Processing and Image Science for the *Journal of the Optical Society of America—A* and is a member of the Editorial Board for *The International Journal of Neurocomputing*. He is the author of *Introduction to Shannon Sampling and Interpolation Theory* (Springer-Verlag, 1991). He has published and holds patents in the areas of signal analysis, detection theory, signal recovery, optical computing, signal processing, and artificial neural processing. Dr. Marks is a cofounder of the Christian Faculty Fellowship at the University of Washington and is a member of Eta Kappa Nu and Sigma Xi.