[illegible]

LA-UR--90-2358

DE90 014926

Received by OSTI

AUG 0 6 1990

TITLE RECURSIVE LEAST-SQUARES LEARNING ALGORITHMS FOR NEURAL NETWORKS

AUTHOR(S) P. S. Lewis, MEE-3
Jenq-Neg Hwang, U.of WA

SUBMITTED TO SPIE - Advanced Signal Processing Algorithms, Architectures & Implementations,
San Diego, CA, July 10-12, 1990

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

Recursive least-squares learning algorithms for neural networks

Paul S. Lewis
Los Alamos National Laboratory*
Mechanical and Electronic Engineering Division
MS J580, Los Alamos, NM 87545

Jenq-Neng Hwang
University of Washington
Department of Electrical Engineering
FT-10, Seattle, WA 98195

ABSTRACT

This paper presents the development of a pair of recursive least squares (RLS) algorithms for online training of multilayer perceptrons, which are a class of feedforward artificial neural networks. These algorithms incorporate second order information about the training error surface in order to achieve faster learning rates than are possible using first order gradient descent algorithms such as the generalized delta rule. A least squares formulation is derived from a linearization of the training error function. Individual training pattern errors are linearized about the network parameters that were in effect when the pattern was presented. This permits the recursive solution of the least squares approximation, either via conventional RLS recursions or by recursive QR decomposition-based techniques. The computational complexity of the update is $\mathcal{O}(N^2)$, where N is the number of network parameters. This is due to the estimation of the $N \times N$ inverse Hessian matrix. Less computationally intensive approximations of the RLS algorithms can be easily derived by using only block diagonal elements of this matrix, thereby partitioning the learning into independent sets. A simulation example is presented in which a neural network is trained to approximate a two dimensional Gaussian bump. In this example, RLS training required an order of magnitude fewer iterations on average (527) than did training with the generalized delta rule (6,331).

1 BACKGROUND

Artificial neural networks (ANNs) offer an interesting and potentially useful paradigm for nonlinear signal processing and pattern recognition. The majority of ANN applications employ the feed-forward, multilayer perceptron (MLP) network architecture in which network parameters are "trained" by a supervised learning algorithm employing the *generalized delta rule* (GDR) [1, 2]. The GDR algorithm approximates a fixed step steepest descent algorithm using derivatives computed by error *backpropagation*. The GDR algorithm is sometimes referred to as the backpropagation algorithm. However, in this paper we will use the term backpropagation to refer only to the process of computing error derivatives.

While multilayer perceptrons provide a very powerful nonlinear modeling capability, GDR training can be very slow and inefficient. In linear adaptive filtering, the analog of the GDR algorithm is the least mean-squares (LMS) algorithm. Steepest descent-based algorithms such as GDR or LMS are *first order* because they use only first derivative or gradient information about the training error to be minimized. To speed up the training process, *second order* algorithms may be employed that take advantage of second derivative or Hessian matrix information.

Second order information can be incorporated into MLP training in different ways. In many applications, especially in the area of pattern recognition, the training set is finite. In these cases, block learning can be applied using standard nonlinear optimization techniques [3, 4, 5]. These techniques iteratively

*Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

minimize the error for the *complete* training set and are based on the ability to evaluate the total error and its derivatives with respect to the network parameters at arbitrary values of the parameters. In essence, the training set is rerun for each function and derivative evaluation. Assuming the number of training patterns is proportional to the number of network parameters, the computational complexity of computing the overall training set error and its gradient is $\mathcal{O}(N^2)$, where N is the number of network parameters. The proposed algorithms also have computational complexity of $\mathcal{O}(N^2)$ per iteration and hence are well suited to block training.

In this work, we address the alternate case of recursive or online network training. Recursive training seeks to adjust the network parameters as training patterns are presented, rather than after a complete pass through the training set. This approach is necessary when the training set is infinite or at least orders of magnitude larger than the number of network parameters. In principle, each training pattern is seen only once, as would be the case in a time series filtering operation. In practice, recursive training may be applied to finite training sets by repeated application of the same set of patterns. Underlying this approach is the assumption that the knowledge to be extracted from the training set is distributed and that no single training pattern contains unique information.

In linear adaptive filtering, recursive least squares (RLS) algorithms implement second order recursive training. The basic approach that underlies RLS algorithms for linear adaptive filters may also be applied to the training of feed-forward ANNs in general and MLPs in particular. RLS-like algorithms for MLPs have been derived in a variety of ways and use different amounts of second order information. These derivations include the application of the extended Kalman filtering equations to MLP training [6], neuron-local linearizations of the sigmoid functions [7], and quadratic MLP error approximations [8, 9].

In this paper we present a unified framework in which to develop and analyze RLS-like algorithms for MLP training. An approximate least squares formulation of the training problem is derived from a linearization of the error function, which yields a quadratic squared-error function. We explicitly show what approximations are necessary both for this least squares formulation and for its recursive solution. We further illustrate how the formulated least squares problem can be solved by either conventional RLS recursions or by the more numerically stable QR decomposition-based methods popular in linear least squares filtering. Within this framework, the RLS-like training algorithms proposed to date use conventional RLS recursions and are mainly differentiated by the portions of the Hessian matrix that they utilize.

2 MULTILAYER PERCEPTRONS

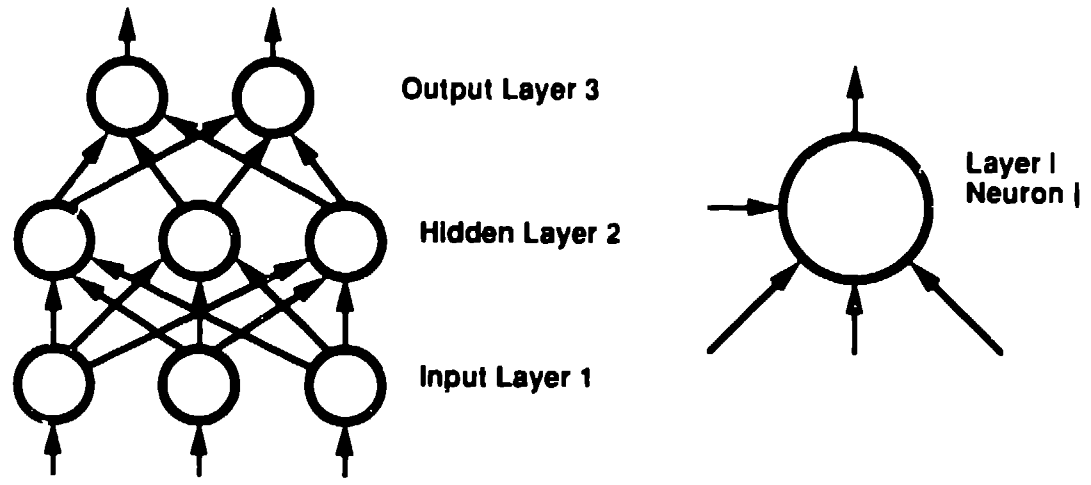
In this work, a multilayer perceptron (MLP) is defined to consist of successive layers of "neurons" interconnected in a feedforward manner. The general structure is illustrated in Fig. 1. The outputs of the neurons in the first layer are the MLP inputs, while the outputs of the neurons in the final layer are the MLP outputs. Neurons in the second and succeeding layers receive the outputs of the previous layer's neurons, generate an affine combination of these values, and run the result through a differentiable nonlinear sigmoid function. Hence, the output of the jth neuron in the lth layer is computed by

$$z_j^l = f\left(c_j^l + \sum_i w_{ij}^l z_i^{l-1}\right) , \tag{1}$$

where the weights w_{ij}^l and offset c_j^l represent the affine combination and $f(\cdot)$ represents the sigmoid function,

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

Since this network has no feedback, the response to an input can be deterministically computed with $\mathcal{O}(N)$ computations, where N is the number of parameters of the MLP. These parameters are the set of weight



n	=	number of training patterns.
N	=	number of network parameters (weights and offsets).
$\underline{\theta}$	=	vector of network parameters .
L	=	number of network layers.
M	=	number of network outputs.

Fig. 1: Multilayer perceptron: (a) Overall network structure. (b) Individual neuron structure.

w_{ij}^{l} and offsets c_j^l for all of the neurons. These can be arranged as an N-dimensional parameter vector $\underline{\theta}$. (In this paper, vectors are column vectors and are denoted by underlines. Matrices are denoted by boldface type.)

In supervised learning, a set of MLP inputs and desired outputs is provided and the purpose of the learning algorithm is to adjust the MLP parameters so that the actual outputs most closely match the desired ones. The goal of learning can be expressed mathematically as the minimization of the total squared error over the training set. Define $e_{kj}(\underline{\theta})$ as the error of the jth MLP output, using network parameters $\underline{\theta}$, in response to the kth training pattern. The total squared error for the complete training set is defined to be

$$E \equiv \frac{1}{2}\sum_k \sum_j \{e_{kj}(\underline{\theta})\}^2 \equiv \sum_k \varepsilon_k(\underline{\theta}) \quad . \tag{3}$$

where ε_k is the total squared error for training pattern k. The gradient with respect to $\underline{\theta}$ is

$$\nabla E = \sum_k \nabla\varepsilon_k(\underline{\theta}) \quad . \tag{4}$$

In multilayer perceptrons with differentiable nonlinearities such as the sigmoid function, $\underline{\nabla}\varepsilon_k(\underline{\theta})$ can be efficiently computed by repeated application of the chain rule. The algorithm to do this requires $O(N)$ computations and is called *backpropagation* [1, 2].

In this paper, we consider recursive or online learning techniques in which parameter adjustments are made based on individual training patterns. The goal is to minimize the error for all training patterns seen, updating the parameters as new training samples become available. This approach is necessary when the training set is effectively infinite, as in the case of filtering operations. In this case, it is also desirable to weight the error function by an exponential window to favor recent training samples over previous ones. To reflect the fact that the network parameters are updated for each additional training pattern, the network parameter vector is indexed to indicate the portion of the training set it is expected to match. Define

- $\underline{\theta}_{n+1}$ as the parameter vector to be computed from training patterns 1 to n, and
- $\varepsilon_{k|m} \equiv \varepsilon_k(\underline{\theta}_m)$ as the squared error generated by training pattern k using parameter vector $\underline{\theta}_m$, and
- λ as an exponential "forgetting" factor between 0 and 1.

The training goal for patterns 1 to n can then be expressed as

$$\min_{\underline{\theta}_{n+1}} E_n \equiv \sum_{k=1}^{n} \lambda^{n-k} \varepsilon_{k|n+1} \ . \tag{5}$$

The gradient of the error function with respect to $\underline{\theta}_{n+1}$ is

$$\Sigma E_n = \sum_{k=1}^{n} \lambda^{n-k} \Sigma \varepsilon_{k|n+1} \ . \tag{6}$$

Minimizing (5) by a fixed step steepest descent approach requires updates of the form

$$\underline{\theta}_{n+1} = \underline{\theta}_n - \mu \Sigma E_n \ . \tag{7}$$

Exact computation of ΣE_n is not possible as it requires knowledge of the unknown parameters $\underline{\theta}_{n+1}$. In *block* learning schemes, the current gradient is approximated by the gradient at the previous parameter values.

$$\Sigma E_n = \sum_{k=1}^{n} \lambda^{n-k} \Sigma \varepsilon_{k|n+1} \approx \sum_{k=1}^{n} \lambda^{n-k} \Sigma \varepsilon_{k|n} \ . \tag{8}$$

This still requires that the current parameters $\underline{\theta}_n$ be applied to all past training patterns. In a recursive learning scheme this is avoided by using a recursive approximation of ΣE_n.

$$\Sigma E_n \approx \dot{\Sigma} E_n \equiv \sum_{k=1}^{n} \lambda^{n-k} \Sigma \varepsilon_{k|k} = \Sigma \varepsilon_{n|n} + \lambda \dot{\Sigma} E_{n-1} \ . \tag{9}$$

This recursive approximation can be interpreted as making a local linear assumption about the error surface so that the gradients are constant $\Sigma \varepsilon_{k|n+1} \approx \Sigma \varepsilon_{k|k}$. Of course, this approximation breaks down as the difference between $n+1$ and k increases, but the exponential forgetting factor λ compensates for this.

Using the recursive gradient approximation in the steepest descent update of (7) yields

$$\begin{aligned} \underline{\theta}_{n+1} &= \underline{\theta}_n - \mu \dot{\Sigma} E_n = \underline{\theta}_n - \mu \left(\Sigma \varepsilon_{n|n} + \lambda \dot{\Sigma} E_{n-1} \right) \\ &= \underline{\theta}_n - \mu \Sigma \varepsilon_{n|n} + \lambda (\underline{\theta}_n - \underline{\theta}_{n-1}) \ . \end{aligned} \tag{10}$$

This is a vector expression for the GDR algorithm with *momentum* [1, 2] where μ is the step size and λ is the momentum constant. For $\lambda = 0$ this reduces to the basic GDR algorithm where each step is based on the instantaneous gradient. The GDR algorithm represents a first order approach to recursive training. Parameter updates are based on first derivative information and the training of each individual parameter is independent of the others. The approach is computationally efficient, requiring only $O(N)$ computations per update, but can converge very slowly for complex error surfaces.

3 LEAST SQUARES FORMULATION

To speed up convergence, second order information can be utilized to indicate the curvature of the error surface. This is achieved by using a quadratic approximation of E_n. A quadratic approximation can be derived by a linearization of the underlying individual errors that comprise E_n.

$$E_n \equiv \frac{1}{2} \sum_{k=1}^{n} \sum_{j} \{\epsilon_{kj|n+1}\}^2 \ . \tag{11}$$

As in the case of approximating the gradient for steepest descent, it is desirable to avoid having to apply current parameters to past training samples. To achieve this, the errors of past training sample k using the parameters $\underline{\theta}_{n+1}$ are linearly approximated by a two term Taylor's series expansion about the errors of sample k using parameters $\underline{\theta}_k$.

$$e_{kj|n+1} \approx e_{kj|k} + \nabla e_{kj|k} \cdot (\underline{\theta}_{n+1} - \underline{\theta}_k) \quad \Longrightarrow \quad \nabla \epsilon_{kj|n+1} \approx \nabla \epsilon_{kj|k} \ . \tag{12}$$

The error $\epsilon_{kj|k}$ is computed by forward evaluation of the MLP for the training pattern k using parameters $\underline{\theta}_k$. The gradient $\nabla e_{kj|k}$ can be computed in a manner analogous to the backpropagation computation of $\nabla \varepsilon$. Details of the procedure are presented in the Appendix.

Applying the Taylor's series approximation in (12) leads to

$$\begin{aligned} \nabla E_n &= \sum_{k=1}^{n} \lambda^{n-k} \nabla \varepsilon_{k|n+1} = \sum_{k=1}^{n} \lambda^{n-k} \sum_j \epsilon_{kj|n+1} \nabla \epsilon_{kj|n+1} \\ &\approx \sum_{k=1}^{n} \lambda^{n-k} \sum_j \nabla \epsilon_{kj|k} \left\{ \epsilon_{kj|k} + \nabla \epsilon_{kj|k} \cdot (\underline{\theta}_{n+1} - \underline{\theta}_k) \right\} \ . \end{aligned} \tag{13}$$

In contrast, a *block* learning scheme would apply the approximations $\epsilon_{kj|n+1} \approx \epsilon_{kj|n} + \nabla \epsilon_{kj|n} \cdot (\underline{\theta}_{n+1} - \underline{\theta}_n)$ and $\nabla \epsilon_{kj|n+1} \approx \nabla \epsilon_{kj|n}$. This leads to the standard Gauss-Newton result for nonlinear least squares which specifies the parameter update: $\nabla E_n \approx \sum_{k=1}^{n} \lambda^{n-k} \sum_j \nabla \epsilon_{kj|n} \left\{ \epsilon_{kj|n} + \nabla \epsilon_{kj|n} \cdot (\underline{\theta}_{n+1} - \underline{\theta}_n) \right\}$.

Defining

$$\nabla_{kj} \equiv \nabla \epsilon_{kj|k} \quad \text{and} \quad d_{kj} \equiv \nabla_{kj}^T \underline{\theta}_k - \epsilon_{kj|k} \ , \tag{14}$$

where the superscript T denotes vector transpose, and setting $\nabla E_n = \underline{0}$ in (13) leads to the following set of normal equations.

$$\left(\sum_{k=1}^{n} \lambda^{n-k} \sum_j \nabla_{kj} \nabla_{kj}^T \right) \underline{\theta}_{n+1} = \sum_{k=1}^{n} \lambda^{n-k} \sum_j \nabla_{kj} d_{kj} \ . \tag{15}$$

To put this into matrix notation, denote the number of MLP outputs by M and define the *Jacobian* matrix $\mathbf{y}_k$, error vector $\underline{\epsilon}_k$, and target vector $\underline{d}_k$.

$$\begin{aligned} \mathbf{y}_k &\equiv [\nabla_{k1} \nabla_{k2} \cdots \nabla_{kM}] & (N \times M) \ , \\ \underline{\epsilon}_k^T &\equiv \left[\epsilon_{k1|k} \, \epsilon_{k2|k} \cdots \epsilon_{kM|k}\right] & (1 \times M) \ , \\ \underline{d}_k^T &\equiv [d_{k1} \, d_{k2} \cdots d_{kM}] = \underline{\theta}_k^T \mathbf{y}_k - \underline{\epsilon}_k^T & (1 \times M) \ . \end{aligned} \tag{16}$$

Using these defintions, (15) can be rewritten as

$$\left(\sum_{k=1}^{n} \lambda^{n-k} \mathbf{y}_k \mathbf{y}_k^T \right) \underline{\theta}_{n+1} = \sum_{k=1}^{n} \lambda^{n-k} \mathbf{y}_k \underline{d}_k \ . \tag{17}$$

Defining block matrix $\mathbf{Y}_k$ and block vector $\underline{D}_k$ as

$$\begin{aligned} \mathbf{Y}_n^T &\equiv \left[\lambda^{\frac{n-1}{2}}\mathbf{y}_1 \ \lambda^{\frac{n-2}{2}}\mathbf{y}_2 \cdots \lambda^0\mathbf{y}_n\right] \quad (N \times nM) \ , \text{ and} \\ \underline{D}_n^T &\equiv \left[\lambda^{\frac{n-1}{2}}\underline{d}_1^T \ \lambda^{\frac{n-2}{2}}\underline{d}_2^T \cdots \lambda^0\underline{d}_n^T\right] \quad (1 \times nM) \ , \end{aligned} \tag{18}$$

permits (15) to be further rewritten as

$$\mathbf{Y}_n^T\mathbf{Y}_n\underline{\theta}_{n+1} = \mathbf{Y}_n^T\underline{D}_n \ . \tag{19}$$

Equation (19) can be solved for $\underline{\theta}_{n+1}$ when $\mathbf{Y}_n$ is full rank. The solution is equivalent to the least squares solution of

$$\mathbf{Y}_n\underline{\theta}_{n+1} = \underline{D}_n \ . \tag{20}$$

The $\mathbf{Y}_n^T\mathbf{Y}_n$ correlation matrix on the left hand side of (19) can be interpreted as an approximation of the second derivative Hessian matrix of the error E_n with respect to the N networks parameters in $\underline{\theta}_n$ [10, 11].

$$\mathbf{Y}_n^T\mathbf{Y}_n = \sum_{k=1}^{n}\lambda^{n-k}\mathbf{y}_k\mathbf{y}_k^T \approx \sum_{k=1}^{n}\sum_{j=1}^{M}\begin{bmatrix} \frac{\partial e_{kj}}{\partial\theta_{n1}}\frac{\partial e_{kj}}{\partial\theta_{n1}} & \cdots & \frac{\partial e_{kj}}{\partial\theta_{n1}}\frac{\partial e_{kj}}{\partial\theta_{bN}} \\ \vdots & & \vdots \\ \frac{\partial e_{kj}}{\partial\theta_{nN}}\frac{\partial e_{kj}}{\partial\theta_{n1}} & \cdots & \frac{\partial e_{kj}}{\partial\theta_{nN}}\frac{\partial e_{kj}}{\partial\theta_{nN}} \end{bmatrix} \approx \begin{bmatrix} \frac{\partial^2 E_n}{\partial^2\theta_{n1}} & \cdots & \frac{\partial^2 E_n}{\partial\theta_1\partial\theta_{nN}} \\ \vdots & & \vdots \\ \frac{\partial^2 E_n}{\partial\theta_N\partial\theta_{n1}} & \cdots & \frac{\partial^2 E_n}{\partial^2\theta_{nN}} \end{bmatrix} \ . \tag{21}$$

So the least squares formulation can be interperted in terms of several levels of approximation. First, as in all second order nonlinear optimization approachs, the nonlinear error surface is approximated by a quadratic [10, 11]. This leads to an expression for the minimum in terms of the gradient and Hessian. Next, as in the Gauss-Newton nonlinear least squares approach, the Hessian matrix is approximated by a sum of products of Jacobian matrices, each of which contains the gradients of the errors of the individual training patterns. Finally, to permit recursive computation, each individual training pattern gradient is approximated by its value at the parameters in effect when the pattern was presented to the network.

4 CONVENTIONAL RLS ALGORITHMS

Equation (19) may be solved recursively using the conventional recursive least squares (RLS) algorithm [12, 13]. The RLS algorithm is based on a pair of recursions for computing $\underline{\theta}_n$ and $\mathbf{P}_n \equiv (\mathbf{Y}_n^T\mathbf{Y}_n)^{-1}$. These recursions are

$$\mathbf{P}_n = \frac{1}{\lambda}\left\{\mathbf{P}_{n-1} - \mathbf{P}_{n-1}\mathbf{y}_n\left(\mathbf{y}_n^T\mathbf{P}_{n-1}\mathbf{y}_n + \lambda\mathbf{I}\right)^{-1}\mathbf{y}_n^T\mathbf{P}_{n-1}\right\} \text{ and} \tag{22}$$

$$\underline{\theta}_{n+1} = \underline{\theta}_n + \mathbf{P}_{n-1}\mathbf{y}_n\left(\mathbf{y}_n^T\mathbf{P}_{n-1}\mathbf{y}_n + \lambda\mathbf{I}\right)^{-1}\left(\underline{d}_n - \mathbf{y}_n^T\underline{\theta}_n\right) \ . \tag{23}$$

These recursions may be simplified by defining the Kalman gain $\mathbf{K}_n \equiv \mathbf{P}_{n-1}\mathbf{y}_n\left(\mathbf{y}_n^T\mathbf{P}_{n-1}\mathbf{y}_n + \lambda\mathbf{I}\right)^{-1}$, the intermediate quantity $\mathbf{x}_n \equiv \mathbf{P}_{n-1}\mathbf{y}_n$, and recalling from (16) that $\underline{d}_n - \mathbf{y}_n^T\underline{\theta}_n = -\underline{\varepsilon}_n$. The general form of the conventional RLS algorithm for the training of MLPs is listed in Table I.

Computation of the error vector $\underline{\varepsilon}_n$ is accomplished by forward evaluation of the MLP and requires $\mathcal{O}(N)$ computations. Computation of the $N \times M$ Jacobian derivative matrix $\mathbf{y}_n$ is accomplished by a backpropagation sweep for each individual output error and requires $\mathcal{O}(NM)$ computations. The matrix inversion in each step is $M \times M$ and requires $\mathcal{O}(M^3)$ computations. For the case of single output networks ($M = 1$), the matrix inversion reduces to scalar division and only a single backpropagation sweep is

TABLE I: ANN RLS ALGORITHM

```
Initialize:
    $\mathbf{P}_0 = \delta^{-1}\mathbf{I}$,    ($\delta$ = small positive constant)
    $\boldsymbol{\ell}_1$ = vector of small random initial weights/offsets

For $n = 1, 2, \cdots$
    Compute $\boldsymbol{\varepsilon}_n$ via MLP forward evaluation.
    Compute $\mathbf{y}_n$ via MLP backpropagation.
    $\mathbf{x}_n = \mathbf{P}_{n-1}\mathbf{y}_n$
    $\mathbf{K}_n = \mathbf{x}_n\left(\mathbf{y}_n^T\mathbf{x}_n + \lambda\mathbf{I}\right)^{-1}$
    $\mathbf{P}_n = \frac{1}{\lambda}\left(\mathbf{P}_{n-1} - \mathbf{K}_n\mathbf{x}_n^T\right)$
    $\boldsymbol{\ell}_{n+1} = \boldsymbol{\ell}_n - \mathbf{K}_n\boldsymbol{\varepsilon}_n$
```

required. The remainder of the update operations require $O(MN^2)$ computations, so overall complexity of the update is $O(MN^2 + M^3)$.

In cases where there is a large number of outputs, the necessity of performing rank M block updates can be avoided by using a scalar root-mean-square (RMS) error in place of the M-dimensional error vector $\boldsymbol{\varepsilon}_n$. This approach permits the use of more computationally efficient rank one updates at the expense of a poorer approximation provided by the least squares formulation. To use this approach, replace the error row vector $\boldsymbol{\varepsilon}_n$ with its scalar RMS value $e_n \equiv |\boldsymbol{\varepsilon}_n| = \sqrt{2\mathcal{E}_n}$. Error minimization now reduces to the single output case and $\nabla e_n = e_n^{-1}\nabla\mathcal{E}_n$ can be computed by a single backpropagation sweep. The computational complexity of the update in this approach is $O(N^2)$.

5 QR-BASED ALGORITHMS

The conventional RLS algorithm presented above recursively solves the normal equations of (19).

$$\mathbf{Y}_n^T\mathbf{Y}_n\boldsymbol{\ell}_{n+1} = \mathbf{Y}_n^T\underline{D}_n \quad \Longrightarrow \quad \boldsymbol{\ell}_{n+1} = \left(\mathbf{Y}_n^T\mathbf{Y}_n\right)^{-1}\mathbf{Y}_n^T\underline{D}_n \ . \tag{24}$$

As noted earlier, this is the least squares solution to (20).

$$\mathbf{Y}_n\boldsymbol{\ell}_{n+1} = \underline{D}_n \ . \tag{25}$$

In the numerical solution of least squares problems, direct solution of the normal equations, as in (24), is rarely used because of the numerical conditioning of the $\mathbf{Y}_n^T\mathbf{Y}_n$ matrix [14]. Instead, the most common approach is to work with the QR decomposition of the data matrix $\mathbf{Y}_n$ in (25). To outline the basic idea, the QR decomposition of the $nM \times N$ matrix $\mathbf{Y}_n$ is

$$\mathbf{Y}_n = \mathbf{Q}_n^T\begin{bmatrix} \mathbf{R}_n \\ \mathbf{0} \end{bmatrix} \ . \tag{26}$$

where $\mathbf{Q}_n^T$ is an $nM \times nM$ orthogonal matrix and $\mathbf{R}_n$ is an $N \times N$ upper triangular matrix. Using this decomposition, the least squares solution of (25) can be derived as follows.

$$\mathbf{Q}_n\mathbf{Y}_n\boldsymbol{\ell}_{n+1} = \mathbf{Q}_n\underline{D}_n \ . \tag{27}$$

TABLE II: ANN QR-RLS ALGORITHM

```
Initialize:
    $\mathbf{R}_0 = 0$
    $\underline{\theta}_1 =$ vector of small random initial weights/offsets
For $n = 1, 2, \cdots$
    Compute $\underline{z}_n$ via MLP forward evaluation.
    Compute $\mathbf{y}_n$ via MLP backpropagation.
    $\underline{d}_n = \mathbf{y}_k^T \underline{\theta}_n - \underline{z}_n$
    $\begin{bmatrix} \sqrt{\lambda}\mathbf{R}_{n-1} & \sqrt{\lambda}\underline{z}_{n-1} \\ \mathbf{y}_n^T & \underline{d}_n \end{bmatrix} \overset{GR}{\Rightarrow} \begin{bmatrix} \mathbf{R}_n & \underline{z}_n \\ 0 & \cdot \end{bmatrix}$ (update via Givens rotations)
    $\underline{\theta}_{n+1} = \mathbf{R}_n^{-1}\underline{z}_n$ (via back substitution)
```

$$\begin{bmatrix} \mathbf{R}_n \\ 0 \end{bmatrix} \underline{\theta}_{n+1} = \begin{bmatrix} \underline{z}_n \\ \underline{v}_n \end{bmatrix} . \tag{28}$$

$$\underline{\theta}_{n+1} = \mathbf{R}_n^{-1}\underline{z}_n . \tag{29}$$

Here $\underline{z}_n$ denotes the first N rows of the product $\mathbf{Q}_n\underline{D}_n$. Since $\mathbf{R}_n$ is upper triangular, (29) may be computed by backsubstitution, which has computational complexity of $\mathcal{O}(N^2)$.

The quantities $\mathbf{Y}_n$ and $\underline{D}_n$ can be expressed recursively as

$$\mathbf{Y}_n = \begin{bmatrix} \sqrt{\lambda}\mathbf{Y}_{n-1} \\ \mathbf{y}_n^T \end{bmatrix} \quad \text{and} \quad \underline{D}_n \begin{bmatrix} \sqrt{\lambda}\underline{D}_{n-1} \\ \underline{d}_n \end{bmatrix} . \tag{30}$$

As a consequence, the QR decomposition which produces $\mathbf{R}_n$ and $\underline{z}_n$ may be computed recursively by [12, 14]

$$\bar{\mathbf{Q}}_n \begin{bmatrix} \sqrt{\lambda}\mathbf{R}_{n-1} & \sqrt{\lambda}\underline{z}_{n-1} \\ \mathbf{y}_n^T & \underline{d}_n \end{bmatrix} = \begin{bmatrix} \mathbf{R}_n & \underline{z}_r \\ 0 & \bullet \end{bmatrix} , \tag{31}$$

where $\bar{\mathbf{Q}}_n$ is an orthogonal matrix that nullifies the elements of $\mathbf{y}_n^T$ by rotating them into the elements of $\sqrt{\lambda}\mathbf{R}_{n-1}$. The matrix $\bar{\mathbf{Q}}_n$ is not computed explicitly, but instead represents a series of Givens rotations [14, 12] that zero the rows containing $\mathbf{y}_n^T$. The form of $\bar{\mathbf{Q}}_n$ is $\bar{\mathbf{Q}}_n = \prod_{i=1}^{M} \prod_{j=1}^{N} \bar{\mathbf{Q}}_n^{(ij)}$, where each matrix $\bar{\mathbf{Q}}^{(ij)}$ represents the Givens rotation that zeros the ijth element of $\mathbf{y}_n^T$ by rotation with the jth row of $\sqrt{\lambda}\mathbf{R}_{n-1}$. The QR-based algorithm for MLP training is listed in Table II. The computational complexity of the update in this approach is $\mathcal{O}(MN^2)$.

6 BLOCK DIAGONAL APPROXIMATIONS

Complete RLS-based MLP training algorithms recursively solve the complete $N \times N$ set of normal equations in (19). For networks with a large number parameters, the $\mathcal{O}(N^2)$ update required for the complete solution can become quite costly. In addition, the update computations are global in nature, combining information from all pairs of parameters. To reduce both the complexity and globality of the required computations, approximate RLS algorithms have been proposed [8, 9, 7]. These algorithms replace the $\mathbf{y}_k\mathbf{y}_k^T$ matrices in (17) with an approximation in which only block diagonal elements of the matrix are retained while the other parts are set to zero. The diagonal blocks retained may correspond to parameters in distinct layers, to the parameters of individual neurons, or even to individual parameters for a straight diagonal approximation.

To develop this interpretation, we rewrite the normal equations (17) that serve as a basis for the RLS algorithms.

$$14z \left(\sum_{k=1}^{n} \lambda^{n-k} \mathbf{y}_k \mathbf{y}_k^T\right) \underline{\theta}_{n+1} = \sum_{k=1}^{n} \lambda^{n-k} \mathbf{y}_k \mathbf{y}_k^T \underline{\theta}_k - \sum_{k=1}^{n} \lambda^{n-k} \mathbf{y}_k \underline{\epsilon}_k \ . \tag{32}$$

From this it is obvious that using a block diagonal approximation for the $\mathbf{y}_k \mathbf{y}_k^T$ matrices on both sides of the equality will decouple the equations into independent sets. Each block set may then be solved independently with either of the RLS or QR-RLS algorithms presented above.

For example, if each block corresponds to the parameters of an individual neuron, then the network parameter vector may be expressed in block form as

$$\underline{\theta} = [\cdots \underline{\theta}^T(\gamma) \cdots]^T \ , \tag{33}$$

where $\underline{\theta}(\gamma)$ represents the parameters (weights and offsets) of the γ neuron. The gradients will also have the same block form

$$\underline{\Sigma}\epsilon_{kj|k} = [\cdots \underline{\Sigma}^T(\gamma)\epsilon_{kj|k} \cdots]^T \ . \tag{34}$$

Following the notation of (14), define

$$\underline{\Sigma}(\gamma)_{kj} \equiv \underline{\Sigma}(\gamma)\epsilon_{kj|k} \quad \text{and} \quad d(\gamma)_{kj} \equiv \underline{\Sigma}^T(\gamma)_{kj}\underline{\theta}(\gamma)_k - \epsilon_{kj|k} \ . \tag{35}$$

This leads to the set of normal equations for $\underline{\theta}(\gamma)$ analogous to (15).

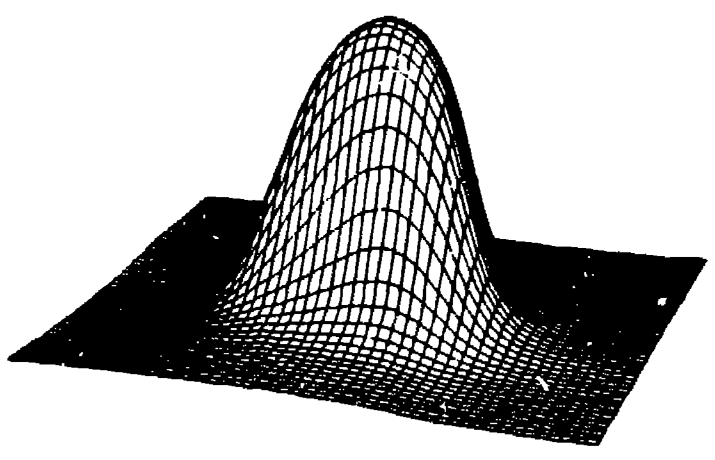
$$\left(\sum_{k=1}^{n} \lambda^{n-k} \sum_{j} \underline{\Sigma}(\gamma)_{kj} \underline{\Sigma}^T(\gamma)_{kj}\right) \underline{\theta}(\gamma)_{n+1} = \sum_{k=1}^{n} \lambda^{n-k} \sum_{j} \underline{\Sigma}(\gamma)_{kj}\, d(\gamma)_{kj} \ . \tag{36}$$

These recursions may be solved recursively by either the conventional RLS algorithm of Table I or the QR algorithm of Table II. In this example, there would be a separate set of recursions for each neuron in the network.

7 SIMULATION EXAMPLE

In this example, a three-layer MLP was used to approximate a two dimensional Gaussian with a standard deviation of one half. (Zero mean, covariance matrix = .25I.) The network had two input neurons, corresponding to the x and y input values, four hidden neurons in the middle layer, and a single neuron in the output layer. To generate a nonrepeating training set, sequential x-y training samples were drawn randomly from the interval -2 to +2. Network weights were initialized to random values between -1 and +1. The network was trained using both the GDR algorithm and the RLS algorithm of Table I. An exponential window of $\lambda = .98$ was used for both algorithms and a step size of $\mu = .1$ was used in the GDR algorithm. Both algorithms were run until either the total squared error (E_n) was less than .1 or until the number of training samples (n) exceeded 25,000.

Fig. 2(a) illustrates a typical result achieved when the error converged to less than .1. Here the forward transfer function of the trained network in plotted for x-y between -2 and 2. The result is easily recognizable as an approximation of a Gaussian. Fig. 2(b) documents the training results for ten different sets of random



(a)

Set	RLS	GDR
1	495	3,437
2	597	6,847
3	505	12,464
4	556	-
5	562	8,811
6	780	-
7	354	2,517
8	474	-
9	478	3,907
10	471	-
mean	527	6,331
standard deviation	105	3484

(b)

Fig. 2: Two dimensional Gaussian approximation simulation. (a) Typical Gaussian approximation. (b) Iterations (training samples) needed to reach an error of .1 for ten different sets of random initial weights.

initial weights. Using the RLS algorithm, training averaged 527 iterations with a standard deviation of 105 iterations. Using the GDR algorithm, training only converged for 6 out of the 10 starting values. In the cases in which training was successful, the average number of iterations required was 6331, with a standard deviation of 3484. In this case, RLS training proved to be more robust than GDR training and on average required an order of magnitude fewer training iterations (samples).

APPENDIX

The least squares algorithms described in this paper use the Jacobian matrices which specify the derivatives of the individual output errors with respect to the network parameters. This is in contrast to more general methods of nonlinear optimization, such as the GDR algorithm, which require only the derivatives of the overall squared error sum with respect to the network parameters. In this appendix we expand on this difference and illustrate how the Jacobian matrices **y** defined in (16) can be computed in a backpropagation-like manner. Figure 3 illustrates the basic network terminology for an L-layer MLP.

The parameters of this network are the weights and offsets, denoted by w^l_{ij}. The vector containing these parameters is $\underline{\theta} \equiv [\cdots\ w^l_{ij}\ \cdots]^T$. For a given training pattern, the relationship between the overall squared error ε and the individual output errors e_m is

$$\varepsilon \equiv \frac{1}{2}\sum_m (e_m)^2 \quad \text{and} \quad \underline{\nabla}\varepsilon = \sum_m e_m \underline{\nabla} e_m \ . \tag{37}$$

The derivatives that compose the gradient of the overall error, $\underline{\nabla}\varepsilon = \partial\varepsilon/\partial\underline{\theta} = \left[\cdots\ \partial\varepsilon/\partial w^l_{ij}\ \cdots\right]^T$ can be found by repeated application of the chain rule [1] where $\delta^l_j \equiv -\partial\varepsilon/\partial a^l_j$.

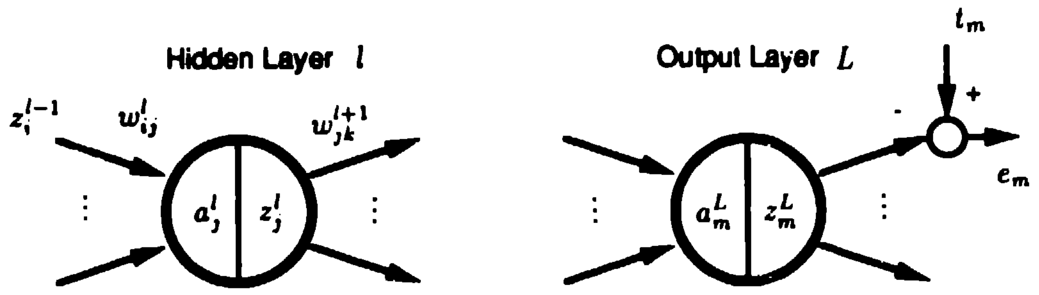


Fig. 3: Multilayer perceptron network terminology.

$$\frac{\partial \varepsilon}{\partial w_{ij}^l} = -\delta_j^l z_i^{l-1} \quad \text{and} \tag{38}$$

$$\delta_j^l = \begin{cases} e_j z_j^L(1-z_j^L) & \text{for } l = L \\ \left(\sum_k \delta_k^{l+1} w_{jk}^{l+1}\right) z_j^l(1-z_j^l) & \text{for } l \neq L \end{cases} \quad . \tag{39}$$

A similar approach can be used to compute the derivatives that compose the gradients of the individual errors $\nabla e_m = \partial e_m/\partial \underline{\theta} = \left[\cdots \partial e_m/\partial w_{ij}^l \cdots\right]^T$. Defining $\delta_{m,j}^l \equiv -\partial e_m/\partial a_j^l$, the result is

$$\frac{\partial e_m}{\partial w_{ij}^l} = -\delta_{m,j}^l z_i^{l-1} \quad \text{and} \tag{40}$$

$$\delta_{m,j}^l = \begin{cases} e_j z_j^L(1-z_j^L) & \text{for } l = L, m = j \\ 0 & \text{for } l = L, m \neq j \\ \left(\sum_k \delta_{m,k}^{l+1} w_{jk}^{l+1}\right) z_j^l(1-z_j^l) & \text{for } l \neq L \end{cases} \quad . \tag{41}$$

The Jacobian matrix $\mathbf{y} = [\nabla e_1 \cdots \nabla e_M]$ has one column for each of the M MLP outputs. Hence M distinct backpropagation sweeps are necessary for its computation.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representation by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volume 1: Foundations*, chapter 8. MIT Press, Cambridge, MA, 1986.

[2] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4–22, April 1987.

[3] R. L. Watrous. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. In *Proc. 1987 IEEE Int. Conf. Neural Networks: Vol. II*, pages 619–627, San Diego, CA, June 1987.

[4] I. F. Shepanski. Fast learning in artifical neural systems: Multilayer perceptron training using optimal estimation. In *Proc. 1988 IEEE Int. Conf. Neural Networks: Vol. I*, pages 465–472, San Diego, CA, July 1988.

[5] A. J. Owens and D. L. Filkin. Efficient training of the back propgation network by solving a system of stiff ordinary differential equations. In *Proc. 1989 Int. Joint Conf. Neural Networks: Vol. II*, pages 381–386, Washington D.C., June 1989.

[6] S. Singhal and L. Wu. Training multilayer perceptrons with the extended Kalman algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 133–140. Morgan Kaufmann, San Mateo, CA, 1989.

[7] M. R. Azimi-Sadjadi, S. Citrin, and S. Sheedvash. Supervised learning process of multi-layer perceptron neural networks using fast least squares. In *Proc. 1990 Int. Conf. Acoustics, Speech and Signal Processing*, pages 1381–1384, Albuquerque, NM, April 1990. IEEE.

[8] S. Kollias and D. Anastassiou. Adaptive training of multilayer neural networks using a least squares estimation technique. In *Proc. 1988 IEEE Int. Conf. Neural Networks: Vol. I*, pages 383–389, San Diego, CA, July 1988.

[9] S. Kollias and D. Anastassiou. An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Trans. Circuits and Systems*, CAS-36(8):1092–1101, August 1989.

[10] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, New York, NY, 1986.

[11] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, New York, NY, 1981.

[12] S. Haykin. *Adaptive Filter Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[13] J. M. Mendel. *Lessons in Digital Estimation Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[14] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 1983.