SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 693–714, May 1993

013

ILL-CONDITIONING IN NEURAL NETWORK TRAINING PROBLEMS*

S. SAARINEN†, R. BRAMLEY†, AND G. CYBENKO†‡

Abstract. The *training* problem for feedforward neural networks is nonlinear parameter estimation that can be solved by a variety of optimization techniques. Much of the literature on neural networks has focused on variants of gradient descent. The training of neural networks using such techniques is known to be a slow process with more sophisticated techniques not always performing significantly better. This paper shows that feedforward neural networks can have ill-conditioned Hessians and that this ill-conditioning can be quite common. The analysis and experimental results in this paper lead to the conclusion that many network training problems are ill conditioned and may not be solved more efficiently by higher-order optimization methods. While the analyses used in this paper are for completely connected layered networks, they extend to networks with sparse connectivity as well. The results suggest that neural networks can have considerable redundancy in parameterizing the function space in a neighborhood of a local minimum, independently of whether or not the solution has a small residual.

Key words. neural network training

AMS(MOS) subject classifications. 65F35, 65K99

1. Introduction.

Some neural network techniques are, in a strictly mathematical sense, an approach to function approximation. As with most approximation methods, they require the estimation of certain (possibly nonunique) parameters which are defined by the problem to be solved [14]. In neural network terminology, finding those parameters is called the *training problem*, and algorithms for finding them are called *training algorithms*. This nomenclature comes from analogy with biological systems, since a set of inputs to the function to be approximated are presented to the network, and the parameters are adjusted to make the output of the network close in some sense to the known value of the function.

Feedforward neural networks use a specific parameterized functional form to approximate a desired input/output relation. Typically, a system is sampled resulting in a finite set of pairs $(t, \tau) \in \mathbb{R}^p \times \mathbb{R}$ where the first coordinate is a position in p-dimensional space and the second coordinate refers to the assigned value for the point. The feedforward neural network function, also from $\mathbb{R}^p \mapsto \mathbb{R}$, has a set of parameters, called weights, which must be determined so that the input and output values as given by the sample data are matched as closely as possible by the approximating neural network. The neural network function for the ith input pattern $(i = 1, 2, \ldots, m)$ can be written succinctly in the form

$$F = F(t_i, \mathbf{x}), \tag{1}$$

where t_i is a p-dimensional input vector, and where $\mathbf{x} \in \mathbb{R}^n$ is the weight vector of parameters to be determined. The form of the function $F(t_i, \mathbf{x})$ for a feedforward network is presented and derived in §3. Let the desired value (or output) for the ith input be denoted by τ_i. A common formulation of the training problem is to find

* Received by the editors February 11, 1991; accepted for publication (in revised form) May 18, 1992.

† Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

‡ This author's work was supported in part by National Science Foundation grant MIP-89-11025, Air Force Office of Scientific Research/Defense Advanced Research Projects Agency contract 89-0536, CCR 900000N, and Department of Energy grant DE-FG02-85ER25001.

values of $\mathbf{x}$ such that the norm of $f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))^T$ is minimized, where f_i denotes the residual between the value of the approximating function and the desired value

$$f_i(\mathbf{x}) = F(t_i, \mathbf{x}) - \tau_i.$$

Often the l_2-norm is used in the minimization of such a function, and we obtain a least squares problem

$$||f(\mathbf{x}^*)||^2 = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_i f_i(\mathbf{x})^2. \tag{2}$$

A numerical minimization algorithm is applied to this equation in order to find a suitable $\mathbf{x}^*$. This minimization is called training.

The current interest in using feedforward neural networks for pattern recognition problems has revealed that the training algorithms are computationally time consuming for a large class of algorithms [17]. Examples of these algorithms are backpropagation (which can be classified as a steepest descent algorithm), conjugate gradient algorithms, and other nonlinear optimization algorithms for parameter estimation. However, little research has been directed into the question of understanding why the training algorithms are slow to converge on neural networks even though the underlying techniques often perform very well for other problems. This paper addresses the question of why these algorithms converge slowly by showing that rank-deficiencies may appear in the Jacobian for a neural network, making the problem numerically ill conditioned.

Except for pattern search methods, the convergence properties of optimization algorithms for differentiable functions depend on properties of the first and/or second derivatives of the objective function [9]. For example, steepest descent explicitly requires the first derivative to define its search direction, and implicitly relies on the second derivative whose properties govern the rate of convergence. When optimization algorithms converge slowly (or not at all) for neural network problems, this suggests that the underlying derivative matrices are numerically ill conditioned. The obvious questions to ask in this case are:

- Will using a higher-order method help avoid this problem? For example, will a quasi-Newton or Newton method reduce the number of iterations?
- Is the mathematical formulation of the training problem the "correct" one? For example, by changing the parameterization or scaling of the problem, can the training algorithm be made more rapidly convergent?
- Is the difficulty of solving the training problem an intrinsic feature of the neural network itself, and not an artifact of the problem formulation and chosen training algorithm?

The proliferation of neural network techniques makes it impossible to answer these questions for all types of networks and all types of problems the networks are to solve. This paper concentrates on one type of network, a multilayer feedforward network, and one type of problem, approximating indicator functions of sets in the plane. These choices were made because multilayer feedforward networks are commonly used by researchers and the classification problem is one for which neural nets are potentially suitable; see [3], [15], and [8]. Furthermore, the training problems examined here are primarily *overdetermined*, that is, the number of training data points is greater than or equal to the number of network parameters. This seems to match the intended usage of neural networks, since for problems such as speech recognition or classification

problems the amount of training data available exceeds the number of parameters that can be accommodated in a practical neural network. The results, however, also seem to apply to underdetermined problems.

It is important to observe that the rank-deficiency shown and explained in the subsequent analysis is independent of the size of the residual. Therefore, the fact that a network configuration cannot exactly "solve" a classification problem does not mean that the network parameterization is parsimonious or not redundant. In fact, rank-deficiency is an indication that redundancy does occur locally. As a simple example, consider using m polynomials of degree n or less, $n < m$, to perform a least squares fit on $m + 1$ or more data points in general position: the residual will generically be nonzero, but the function class is inherently overparameterized. This redundancy is usually considered an advantage of neural networks, allowing some degree of fault tolerance.

In this paper we will first review nonlinear least squares problems; second, we will describe further the neural network problem and show general properties of the Jacobian for a feedforward neural network; and finally, we will investigate the Jacobian and Hessian for some examples. The analytic results about the Jacobian show that it can be rank-deficient in certain situations that can be enumerated. We then show that this rank-deficiency actually occurs in a number of experiments. For methods that use Hessian information explicitly, the rank-deficiency or ill-conditioning of the Jacobian still plays a role because the Jacobian part of the Hessian is dominant. We emphasize that this paper deals only with the least squares formulation of a training algorithm, and that all the results obtained are for this class of problems.

2. Review of nonlinear least squares. The material in this section establishes notation and describes the difficulties imposed by a rank-deficient or ill-conditioned Jacobian. We define the rank and the condition number of a matrix $A \in \mathbb{R}^{l\times q}$ $(l \geq q)$ by the singular value decomposition $A = U\Sigma V^T$, where $U^TU = I_l$, $V^TV = I_q$, and $\Sigma \in \mathbb{R}^{l\times q}$ is a diagonal matrix $\text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_q)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_q \geq 0$. A has rank $r < q$ if $\sigma_{r+1} = \cdots = \sigma_q = 0$ and $\sigma_r \neq 0$. The *degree of rank-deficiency* of A is $q - r$. The *condition number* of A is $\kappa = \kappa(A) = \sigma_1/\sigma_r$, and A is *ill conditioned* if $\kappa(A)$ is "large." Except in special cases, in practice it is rare to find singular values exactly equal to zero, and numerically determining r is difficult (see, for example, Figs. 6–8). Because of this the numerical examples of this paper will take $q = r$, but the analytic descriptions will also discuss the $q < r$ case. Note that if $\sigma_{r+1}, \ldots, \sigma_q > 0$ are all very small relative to σ_1, A is close to being rank-deficient of degree at least $q - r$.

Letting $\phi(\mathbf{x}) = \frac{1}{2} \parallel f(\mathbf{x}) \parallel^2$ in (2), the gradient of ϕ is

$$\nabla\phi(\mathbf{x}) = J(\mathbf{x})^T f(\mathbf{x}), \tag{3}$$

and the Hessian matrix of ϕ is

$$H(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) H_i(\mathbf{x}), \tag{4}$$

where $[J_{ij}] = \partial f_i/\partial x_j$ is the Jacobian matrix of $f(\mathbf{x})$ and $H_i(\mathbf{x})$ is the Hessian matrix of the component function $f_i(\mathbf{x})$. For convenience, the explicit dependence on $\mathbf{x}$ will sometimes be omitted by writing $H = J^TJ + \sum_{i=1}^{m} f_i H_i$, etc. Algorithms for minimizing $\phi(\mathbf{x})$ usually take advantage of the special structure of $\nabla\phi(\mathbf{x})$ and $H(\mathbf{x})$; the reader is referred to [5] for a survey of such algorithms.

A given point $\mathbf{x}^*$ is a *critical point* of ϕ if $\nabla\phi(\mathbf{x}^*) = 0$, and $\mathbf{x}^*$ is a local minimum only if all the eigenvalues of $H(\mathbf{x}^*)$ are nonnegative. One of the weaknesses of commonly used neural network training algorithms is the failure to check the obtained "solution" for optimality, except in the trivial consistent case where $f(\mathbf{x}^*) = 0$. Optimization methods for solving the nonlinear least squares problem generate a search direction $\mathbf{p}$ and a stepsize α from the current iterate $\mathbf{x}$. The search direction and stepsize are usually chosen so that $\phi(\mathbf{x}+\alpha\mathbf{p}) < \phi(\mathbf{x})$. Table 1 shows the search directions used by some common optimization methods, assuming that J is full rank.

TABLE 1
Search directions used by various optimization methods.

Algorithm	Search direction
Steepest descent	$-J^T f$
Conjugate gradient	$-J^T f + \beta\tilde{p}$, with $\tilde{p}$ = previous search direction β = a scalar
Newton	$-(J^T J + \sum_{i=1}^{m} f_i H_i)^{-1} J^T f$
Gauss–Newton	$-(J^T J)^{-1} J^T f$
Levenberg–Marquardt	$-(J^T J + \rho_k I)^{-1} J^T f$
Quasi-Newton	$-(J^T J + B_k)^{-1} J^T f$, B_k satisfying a quasi-Newton condition

2.1. Local convergence. The local convergence properties of the methods in Table 1 depend on the size of the residual $f(\mathbf{x}^*)$ and the rank and condition number of $H(\mathbf{x}^*)$, the Hessian at the solution. Steepest descent has a q-linear rate of convergence (see [11] for a definition of q-linear) with an asymptotic error constant proportional to $(\kappa-1)/(\kappa+1)$, where κ is the condition number of $H(\mathbf{x}^*)$ [9]. Conjugate gradient methods generally have a linear rate of convergence, but their behavior depends on the definition of the conjugacy scalar β as well as the frequency of *restart*, i.e., reinitialization of the algorithm [13]. Quasi-Newton methods have a superlinear rate of convergence if, roughly speaking, $H(\mathbf{x}^*)$ is nonsingular and the matrices B_k are chosen so that $J^T J + B_k$ approximates $H(\mathbf{x}^*)$ along the search directions; see [4] for a more detailed description of both quasi-Newton methods and their convergence properties. Newton's method has a quadratic rate of convergence, provided that the Hessian is nonsingular at $\mathbf{x}^*$. The other methods have local convergence properties that can be seen by considering them as approximations to Newton's method. For example, if the residual is zero at $\mathbf{x}^*$, then $H(\mathbf{x}^*) = J^T(\mathbf{x}^*)J(\mathbf{x}^*)$ and the Gauss–Newton method shares the quadratic convergence rate of Newton's method if $J(\mathbf{x}^*)$ is full rank. Under the same conditions the Levenberg–Marquardt method has quadratic convergence when the ρ_k in Table 1 is zero; in practice, ρ_k is chosen to provide better global convergence. When the residual is large, however, Gauss–Newton and Levenberg–Marquardt methods have a linear convergence rate.

2.2. Global convergence. Minimization algorithms can spend the majority of their time in finding a neighborhood of the solution in which local convergence theorems apply, so the local convergence rate is irrelevant if the algorithm fails to enter that neighborhood or if it is extremely small. Note the critical dependence of all of the methods in Table 1 on the Jacobian J; each simply multiplies the *fundamental* search direction $-J^T f$ by some matrix estimating second-order information (for steepest descent the matrix is I) except for the conjugate direction, which has a linear combination of the current and previous vectors $-J^T f$ as its search direction. The

search directions for the Newton or Gauss–Newton methods are ill defined when the Hessian is singular or the Jacobian is not full rank, respectively, but the Levenberg–Marquardt parameter ρ_k and the quasi-Newton matrix B_k are normally chosen to assure that their respective search directions are well defined. Unfortunately, if J or the Hessian are rank-deficient or ill conditioned at many of the iterate points, the methods can actually perform worse than steepest descent, which in turn can fail to converge in finite precision arithmetic.

2.3. Regularization. When J is rank-deficient or ill conditioned, these algorithms can be generalized by two common regularization approaches. The first replaces J with $J_r = U\Sigma_r V^T$, where $\Sigma_r = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0, \ldots, 0)$, $J = U\Sigma V^T$ is the singular value decomposition of J, and r is an estimate of the rank of J as discussed above. However, the computational determination of r is a difficult problem, and can have a dramatic effect on the resulting search direction (see the example in [6, p. 136]). Furthermore, the resulting search direction has nonzero components only in a subspace of dimension r. When $r \ll n$ and the subspace changes slowly from iteration to iteration, the method can fail to make sufficient progress to a minimum. Methods for circumventing this difficulty generally add a component in the orthogonal complement of the subspace of dimension r to the search direction, as in [6]. A second regularization approach adds a small multiple $\gamma \parallel \mathbf{x} \parallel$ of the norm of the weight vector to the objective function, possibly allowing γ to change on each iteration. This restricts the size of the weights, but unfortunately the weights frequently need to be large for accurate learning, i.e., a good least squares solution. So if γ is prevented from decreasing to zero the quality of solution can suffer, while if γ is decreased to zero eventually the same ill-conditioning problems are encountered. Nevertheless, the Levenberg–Marquardt algorithm used in §3.4 implicitly uses this second form of regularization (see [10]), and can provide adequate solutions in many cases.

For most overdetermined nonlinear least squares problems, these considerations are minor. Generally the Jacobian is full rank (but ill-conditioning can occur) and it is only at exceptional points that J is rank-deficient, but even then the rank-deficiency is small. For neural network problems, however, we show that a large number of columns of the Jacobian can easily be nearly linearly dependent and so the matrix is close in 2-norm to a matrix with a large degree of rank-deficiency.

3. Neural network training problems. A neural network consists of three types of computational elements or nodes arranged in layers: input layer nodes, hidden layer nodes, and output layer nodes with weighted connections between them. Each node has *several* inputs and only *one* output. A feedforward neural network has connections between neighboring layers with information flowing only in one direction. There are no connections within a layer. We further use networks which are fully connected between layers, that is, every node in a given layer has a directed connection with all the nodes in the previous layer and in the successive layer. Such a neural network with two hidden layers is depicted in Fig. 1. The flow of information is upwards in the figure. The input layer nodes and the output layer nodes are depicted by triangles, and the hidden layer nodes by circles. All arrows leaving a given node denote the unique output for the node. A hidden layer node forms its output o by first forming the weighted sum of its inputs u_i,

$$s = \sum_{i=1}^{j} x_i u_i + x_0,$$

where the $x_i, i = 0, \ldots, j$ are called weights and x_0 is called an offset. The hidden node then applies a univariate excitation function $\sigma(s)$ to the weighted sum s and this value, $o = \sigma(s)$, becomes the output of the node. The input layer and output layer nodes do not apply an excitation function to their inputs. Therefore, the *inputs* in Fig. 1 are the same as the network inputs (shown as v_1 and v_2). A more elaborate description of neural networks in general can be found in [15]. We will next derive the functional form of the network function.

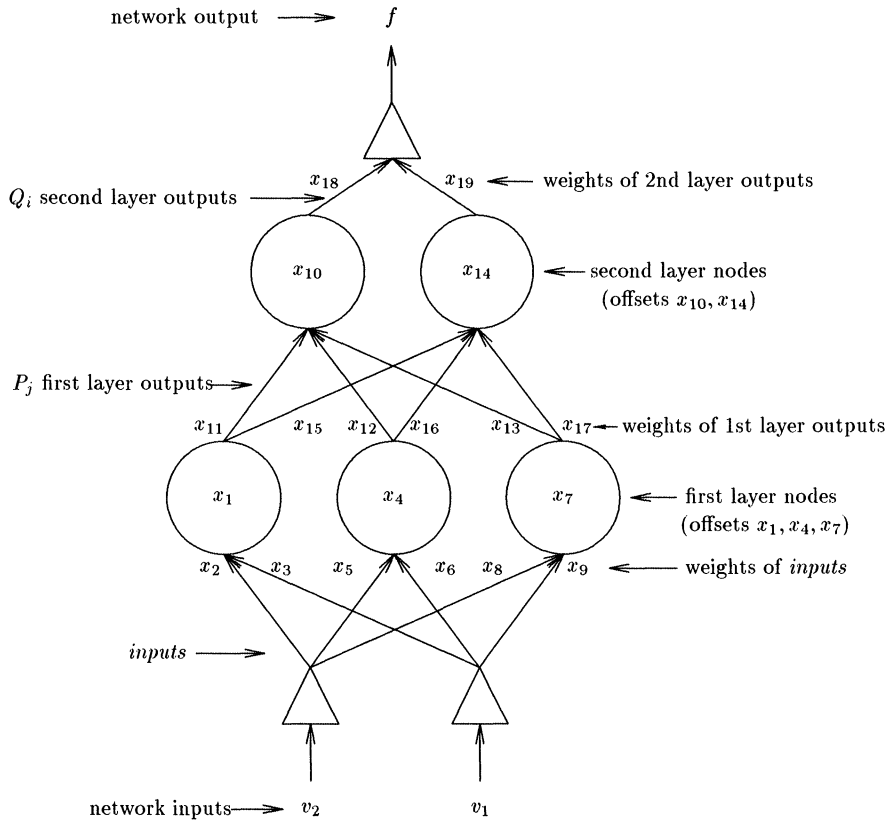

FIG. 1. *A 2-3-2 feedforward neural network with the weights indicated by $x_i, i = 1, \ldots, 19$. The offsets are written inside the nodes (indicated by circles). The quantities P_j and Q_i are discussed in the text.*

Let the number of inputs to the network be h, the number of first layer nodes be p, and the number of second layer nodes be s. The network can then be concisely labeled as $h - p - s$ and Fig. 1 represents a 2-3-2 network. Let the total number of training data points be m and the total number of parameters (i.e., weights associated with the arcs and offsets in the nodes) be $n = p(h + s + 1) + 2s$.

For the ith training data input pattern we write

$$t_i = (v_1^{(i)}, v_2^{(i)}, \ldots, v_h^{(i)}),$$

and let weights associated with the input layer arcs be $x_{(j-1)(h+1)+1+k}$, $1 \le j \le p$ and $1 \le k \le h$, and the offsets associated with the first layer nodes be $x_{(j-1)(h+1)+1}$, where $1 \le j \le p$.

Then we can write for the output of the jth first layer node for the training data point i

$$\sigma(P_j^{(i)}) = \sigma\left(\sum_k x_{(j-1)(h+1)+1+k}\, v_k^{(i)} + x_{(j-1)(h+1)+1}\right), \tag{5}$$

where σ is a sigmoidal or excitation function. Similarly, for the second layer nodes let the weights on the arcs be $x_{t+(l-1)(f+1)+1+j}$, where $t = p(h+1)$, $1 \le l \le s$ and $1 \le j \le p$, and the offsets be $x_{t+(l-1)(p+1)+1}$, where $1 \le l \le s$. The output of the lth second layer node for training data point i is

$$\sigma(Q_l^{(i)}) = \sigma\left(\sum_j x_{t+(l-1)(p+1)+1+j}\, \sigma(P_j^{(i)}) + x_{t+(l-1)(p+1)+1}\right). \tag{6}$$

Furthermore, if the weights on the last layer are x_{u+l}, where $1 \le l \le s$ and $u = t + s(p+1)$, then the output of the network for training data point i is

$$F(t_i, \mathbf{x}) = \sum_l x_{u+l}\, \sigma(Q_l^{(i)}). \tag{7}$$

In this paper we assume that the node on the last layer will not apply an excitation function to its inputs (this will not make the problem less general but rather avoid using one extra parameter). The aim is to find the weights on the arcs and the offsets by minimizing (2) and using some input-output pairs of the patterns that the network should approximate.

The form of the excitation function can vary. Examples include

$$\sigma_1(x) = \frac{1}{1+e^{-x}} \quad \text{and} \quad \sigma_2(x) = \frac{2}{\pi}\tan^{-1} x. \tag{8}$$

The functions $\sigma_1(x)$ and $\sigma_2(x)$ are called sigmoidal functions, and their ranges are $(0,1)$ and $(-1,1)$, respectively. Note that both their derivatives approach 0 when $|x| \gg 1$. The general form of these functions can be obtained by making the transformation $x \mapsto \alpha x + \beta$, but the ranges of the functions and their general form stay the same. It has been shown that under very general conditions on the activation functions, such classes of neural networks as in (7) are universal approximators [3]. The main reason for choosing the two functions in (8) is that they can approximate the hard delimiter step function and are continuously differentiable. In this paper we will use $\sigma_1(x)$, but the method in this paper can be used to derive similar results for other excitation functions as well (including radial basis functions [2], [12]).

3.1. Explicit form of the Jacobian for a two-layer network. The Jacobian for the multilayer feedforward network problem can be written explicitly using the notation from the previous section. The Jacobian is a matrix of size $m \times n$, where each row of the Jacobian corresponds to a single training data point. We present the Jacobian layer by layer and specifically show the components for the arc weights separately from the offsets. Thus we will present a row of the Jacobian, J, but will show the row in blocks (the first subscript refers to the training data point i, and the

TABLE 2
Summary of the Jacobian.

Jacobian element	Formula	Corresponding unknowns
$J_{i,(j-1)(h+1)+1}$	$\sigma'(P_j^{(i)}) \sum_l x_{u+l} x_{t+(l-1)(p+1)+1+j}\, \sigma'(Q_l^{(i)})$	First layer offsets
$J_{i,(j-1)(h+1)+1+k}$	$v_k^{(i)} \sigma'(P_j^{(i)}) \sum_l x_{u+l} x_{t+(l-1)(p+1)+1+j}\, \sigma'(Q_l^{(i)})$	First layer weights
$J_{i,t+(l-1)(p+1)+1}$	$x_{u+l} \sigma'(Q_l^{(i)})$	Second layer offset
$J_{i,t+(l-1)(p+1)+1+j}$	$x_{u+l} \sigma(P_j^{(i)}) \sigma'(Q_l^{(i)})$	Second layer weights
$J_{i,u+l}$	$\sigma(Q_l^{(i)})$	Last layer weights

second to the actual parameter or column of the Jacobian). The form of the resulting Jacobian is summarized in Table 2 and the third column in the table explains the origin of the term.

We can now write the ith row of the Jacobian (using its inherent block structure) for the 2-3-2 network in Fig. 1:

$$
\begin{aligned}
J_{i,1\ldots3} &= [x_{18}x_{11}\sigma'(Q_1^{(i)}) + x_{19}x_{15}\sigma'(Q_2^{(i)})]\sigma'(P_1^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T, \\
J_{i,4\ldots6} &= [x_{18}x_{12}\sigma'(Q_1^{(i)}) + x_{19}x_{16}\sigma'(Q_2^{(i)})]\sigma'(P_2^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T, \\
J_{i,7\ldots9} &= [x_{18}x_{13}\sigma'(Q_1^{(i)}) + x_{19}x_{17}\sigma'(Q_2^{(i)})]\sigma'(P_3^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T, \\
J_{i,10\ldots13} &= x_{18}\sigma'(Q_1^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T, \\
J_{i,14\ldots17} &= x_{19}\sigma'(Q_2^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T, \\
J_{i,18\ldots19} &= (\sigma(Q_1^{(i)}), \sigma(Q_2^{(i)}))^T.
\end{aligned}
$$

Here $J_{i,j}$ denotes the element in the ith row and jth column of the Jacobian. From the dependency on i in each column of J, one row cannot be an exact linear combination of sets of rows. If there are some linear or near-linear dependencies between rows or columns in the Jacobian they must arise in some other manner.

3.2. Properties of the sigmoidal. An explanation of the rank-deficiency of the Jacobian relies on the properties of sigmoidal functions and their derivatives. We consider the sigmoidal function $\sigma(x) = \sigma_1(x)$ in (8), with derivative

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

The columns of the Jacobian for the neural network function has terms of the form $\sigma(x)$, $\sigma'(x)$, $\sigma'(x)\sigma(y)$, and $\sigma'(x)\sigma'(y)$ with constant coefficients. To explain the near linear dependence between columns of J, we are therefore interested in the quantities

$$
\begin{aligned}
\mathrm{A}(x,y) &= \sigma(x+y) - \sigma(x), \qquad \mathrm{B}(x,y) = \sigma'(x+y) - \sigma'(x), \\
\mathrm{C}(x,y) &= \sigma'(x+y)\sigma'(x), \qquad \mathrm{D}(x,y) = \sigma(x+y)\sigma'(x),
\end{aligned}
$$

where $x \in [-\rho, \rho]$ and $y \in [-\delta, \delta]$.

Figure 2 contains the graph for $\mathrm{A}(x,y)$ for various values of $x \in [-10, 10]$ and as a function of $y \in [-20, 20]$. Note that the differences in the graphs for large values of

$|x|$ are small. Figure 3 shows the graph for $B(x, y)$ for $x \in [-10, 10]$ and $y \in [-20, 20]$. Note that $B(x, y)$ varies slowly with x. It is easy to see that $\sup_{x,y} A(x, y) = 1$ and $\sup_{x,y} B(x, y) = 0.25$ from the properties of σ and σ'. Similarly, $|C(x, y)| \leq \frac{1}{16}$ and $|D(x, y)| \leq \frac{1}{4}$, for any x and y. We show the form of these functions in Figs. 4 and 5. The upper limits of these functions are not as important as the fact that often the values of these functions are close to 0 for a large range of x and y.

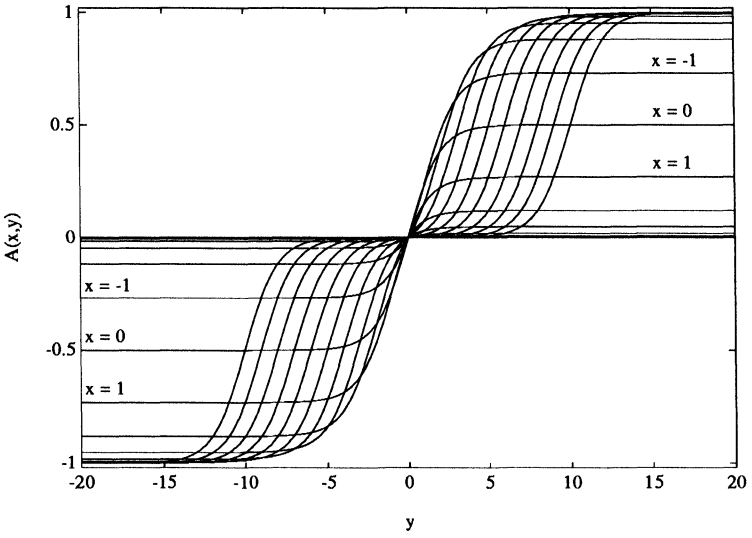

FIG. 2. *Function* $A(x, y)$ *for various* x. *In the figure we have plotted curves for* $x = -10, -9, \ldots, 0, \ldots, 9, 10$, *and three are indicated.*

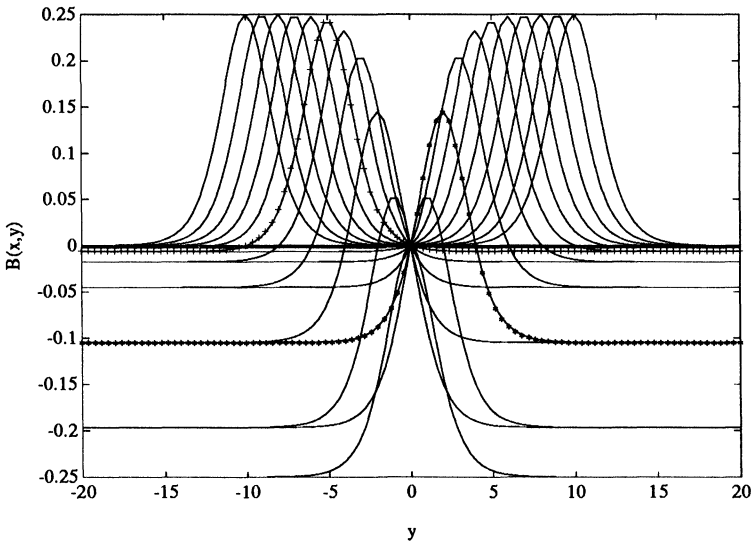

FIG. 3. *Function* $B(x, y)$ *for various* x. *We have used* $x = -10, -9, \ldots, 9, 10$ *and indicated the curves for* $x = 5$ *with* $+$ (*on the left-hand side of the figure*) *and* $x = -2$ *with* $*$ (*on the right-hand side*).

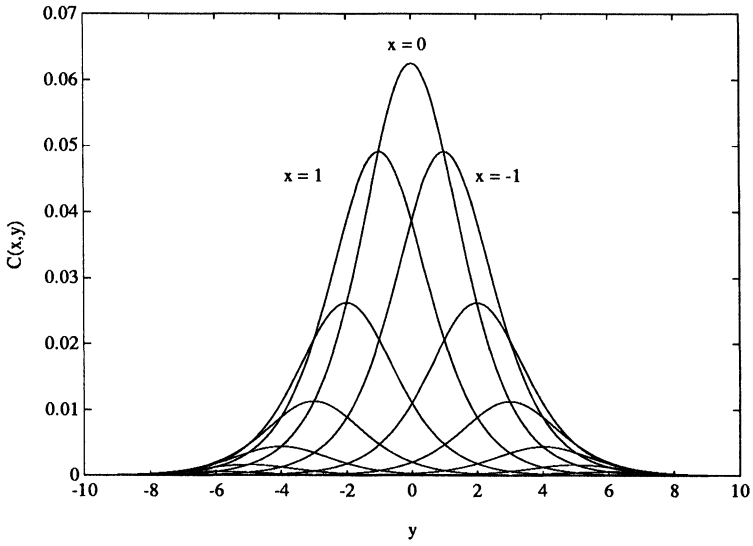

FIG. 4. *Function* $C(x, y)$ *for* $x = -10, -9, \ldots, 9, 10$. *The curves for* $x = -1, 0,$ *and* 1 *are indicated.*

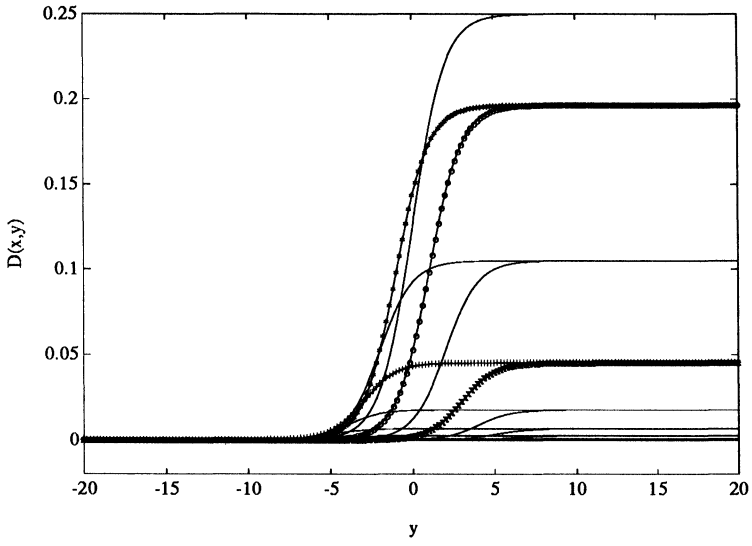

FIG. 5. *Function* $D(x, y)$ *for* $x = -10, -9, \ldots, 9, 10$. *The following curves are indicated:* $x = -1$ (o), $x = 1$ $(*)$, $x = -3$ (x), *and* $x = 3$ $(+)$.

3.3. Rank-deficiency of the Jacobian of a two hidden layer network. This section analyzes five situations where the Jacobian for a two hidden layer feed-forward network is ill conditioned or rank-deficient. These results easily extend to a network with more hidden layers and follow from three simple propositions.

PROPOSITION 3.1. *Let* B *be a submatrix of* A *consisting of columns of* A. *Then* $\kappa(B) \leq \kappa(A)$.

Proof. This follows immediately from a repeated application of [7, Corollary 8.3.3]. □

PROPOSITION 3.2. *Let $A = [x, y] \in \mathbb{R}^{n\times 2}$, and suppose that the angle θ between x and y satisfies $\cos(\theta) = 1 - \epsilon$ for some $\epsilon \in (0, 1)$. Then*

$$\kappa^2(A) \geq \frac{1}{4\epsilon(2-\epsilon)} \cdot \left(\frac{\| y \|^2}{\| x \|^2} + \frac{\| x \|^2}{\| y \|^2} + 2 \right).$$

Proof. As noted before, the singular values of A are the square roots of the eigenvalues of $G = AA^T$. The eigenvalues of G are nonzero and satisfy

$$\lambda = \frac{1}{2}[x^T x + y^T y \pm \sqrt{(x^T x - y^T y)^2 + 4(x^T y)^2}].$$

Using this and completing the square in $\kappa(G) = \lambda_{\max}/\lambda_{\min}$ gives

$$\kappa(G) = \frac{[x^T x + y^T y + \sqrt{(x^T x - y^T y)^2 + 4(x^T y)^2}]^2}{4(x^T x \cdot y^T y - (x^T y)^2)}.$$

Since $\cos(\theta) = \frac{x^T y}{\|x\| \cdot \|y\|} = 1 - \epsilon$, $(x^T y)^2 = (1-\epsilon)^2 x^T x \cdot y^T y$ and so

$$\kappa(G) \geq \frac{(x^T x + y^T y)^2}{4[1-(1-\epsilon)^2] x^T x \cdot y^T y} = \frac{1}{4\epsilon(2-\epsilon)} \cdot \left(\frac{\| y \|^2}{\| x \|^2} + \frac{\| x \|^2}{\| y \|^2} + 2 \right). \qquad \square$$

Proposition 3.2 provides a lower bound on the condition number of a matrix when two columns are nearly collinear. In fact, if some number of columns of a matrix, say p, are almost collinear, then one should expect at least $p-1$ singular values of the matrix to be relatively small. Thus the matrix would almost have a rank-deficiency of $p-1$ in such a case. These claims can be made precise along the lines of Proposition 3.2, but now we assume that the columns are normalized.

PROPOSITION 3.3. *Let $A = [a_1, \ldots, a_p]$ be an $m \times p$ matrix with normalized columns, so that $a_i^T a_j = 1 - \epsilon_{ij}$ for all $1 \leq i, j \leq n$, with $\epsilon_{ii} = 0$ for $i = 1, 2, \ldots, p$. Suppose that $\epsilon_{ij} \leq \epsilon < 1$. Then A has one singular value at least as large as $\sqrt{p(1-\epsilon)}$ while the remaining singular values are no larger than $\sqrt{p\epsilon}$.*

Proof. By assumption, $A^T A = B - E$ where B is the matrix of all ones and $E = [\epsilon_{ij}]$. The eigenvalues of B are p (with multiplicity 1) and 0 (with multiplicity $p-1$), while the eigenvalues of E are no larger than $p\epsilon$ by the Gersgorin disk theorem. An application of the Wielandt–Hoffman theorem [7] shows that $A^T A$ has one eigenvalue λ_p satisfying $|\lambda_p - p| \leq p\epsilon$, and the other eigenvalues λ_i satisfy $|\lambda_i| \leq p\epsilon$. Since the nonzero singular values of A are the square roots of the nonzero eigenvalues of $A^T A$, $\sigma_1 \geq \sqrt{p(1-\epsilon)}$ and $\sigma_i \leq \sqrt{p\epsilon}$ for $i = 2, 3, \ldots, p$. $\square$

The preceding results quantify the following argument. For a Jacobian J to be ill conditioned, it suffices to have two columns that are nearly dependent. If the cosine of the angle between those two columns is $1-\epsilon$, then the condition number of J is at least $\mathcal{O}(\epsilon^{-1/2})$. If p of the columns of J are nearly multiples of each other (that is, the cosine of the angle between any pair of the p vectors satisfies $|\cos\theta| \geq 1-\epsilon$) and the columns are normalized, then at least $p-1$ singular values of J are $\mathcal{O}((p\epsilon)^{1/2})$, and one is $\mathcal{O}(p^{1/2})$. This implies that J is within $p\epsilon$ in the 2-norm to a matrix with degree of rank-deficiency at least $p-1$, which causes difficulties for linear least squares solvers.

To simplify the notation, define

$$\underline{\sigma}(Q_r) = (\sigma(Q_r^{(1)}), \sigma(Q_r^{(2)}), \ldots, \sigma(Q_r^{(m)}))^T$$

and define the vectors $\underline{\sigma}'(Q_r)$, $\underline{\sigma}(P_r)$, and $\underline{\sigma}'(P_r)$ similarly. We now describe five cases where conditions internal to the network lead to ill-conditioned and close to rank-deficient matrices.

Case 1. If for some j, $\underline{\sigma}(P_j)$ is a multiple of $e = (1, 1, \ldots, 1)^T$, then any pair of columns in the Jacobian corresponding to an arc and an offset, which have $\sigma(P_j^{(i)})$ as input, on the second layer will produce two identical columns of J. If this holds for K first layer nodes, then $K+1$ columns of J are identical. Since there are s second layer nodes, this gives a rank-deficiency of sK. Note that this condition is approximately satisfied if the angle between e and $\underline{\sigma}(P_j)$ is small.

Case 2. If $\underline{\sigma}'(Q_k)$ and $\underline{\sigma}'(Q_j)$ are multiples of each other then the block of columns in J corresponding to the parameters of the nodes k and j of the second layer are identical, producing a rank-deficiency of $(p+1)+(h+1)$. Note that this condition is approximately satisfied if the angle between $\underline{\sigma}'(Q_k)$ and $\underline{\sigma}'(Q_j)$ is small. The term $p+1$ appears from the second layer nodes (and parameters associated with them), and the term $h+1$ appears from the Jacobian calculated with respect to first layer parameters. This is shown in (9) for $k=1$ and $j=2$:

$$
\begin{aligned}
J_{i,10\ldots13} &= x_{18}\ \sigma'(Q_1^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T, \\
J_{i,14\ldots17} &= x_{19}\ \sigma'(Q_2^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T.
\end{aligned} \tag{9}
$$

A rank-deficiency of $p+1=4$ occurs because the block array $[e, \underline{\sigma}(P_1), \underline{\sigma}(P_2), \underline{\sigma}(P_3)]^T$ is always identical in $J_{i,10\ldots13}$ and $J_{i,14\ldots17}$. In general, if $\operatorname{rank}[\underline{\sigma}'(Q_1), \underline{\sigma}'(Q_2), \ldots, \underline{\sigma}'(Q_s)] = L$, then the rank-deficiency is $(s-L)(p+1+h+1)$.

Case 3. If $\underline{\sigma}'(P_k)$ and $\underline{\sigma}'(P_j)$ are multiples of each other then the block of columns corresponding to the first layer node k of J (i.e., the arcs and offset parameters) is a multiple of the block of columns corresponding to the first layer node j. Note that this condition approximately holds if $\underline{\sigma}'(P_k)$ and $\underline{\sigma}'(P_j)$ have a small angle between them. As in Case 2 above, the deficiency is $h+1$ because the block array $[e, \underline{v}_1, \underline{v}_2]^T$ is repeated in $J_{i,1\ldots3}$ and $J_{i,4\ldots6}$. More generally, if $\operatorname{rank}[\underline{\sigma}'(P_1), \underline{\sigma}'(P_2), \ldots, \underline{\sigma}'(P_f)] = L$ then the rank-deficiency from this condition is $(p-L)(h+1)$.

Case 4. If $\underline{\sigma}(Q_k)$ and $\underline{\sigma}(Q_j)$ are multiples of each other then the columns corresponding to last layer parameters k and j are linearly dependent. This is shown in (10) for $k=1$ and $j=2$:

$$
J_{i,18\ldots19} = (\sigma(Q_1^{(i)}),\ \sigma(Q_2^{(i)}))^T. \tag{10}
$$

More generally, if $\operatorname{rank}[\underline{\sigma}(Q_1), \underline{\sigma}(Q_2), \ldots, \underline{\sigma}(Q_s)] = L$ then the rank-deficiency is $s-L$.

There is also a possible type of dependency in the columns of the Jacobian similar to Case 4 above, but which in fact rarely occurs.

Case 5. If $\underline{\sigma}(P_k)$ and $\underline{\sigma}(P_j)$ are multiples of each other, but $\underline{\sigma}(P_k)$ is not a multiple of e (so that Case 1 is excluded), then the columns corresponding to the arcs k and j are multiples of each other. More generally, if $\operatorname{rank}[\underline{\sigma}(P_1), \underline{\sigma}(P_2), \ldots, \underline{\sigma}(P_f)] = L$ then the rank-deficiency from this condition is $(p-L)s$.

Finally, note that rank-deficiency can arise in less restrictive ways than indicated by the hypotheses above. For example, in Case 3 it is only necessary that $\sigma'(P_k)$ and $\sigma'(P_j)$ nearly be multiples of each other for those components i with $\sigma'(Q_1^{(i)})$ and $\sigma'(Q_2^{(i)})$ to be nonzero. Because the derivative of the sigmoidal is close to zero for arguments outside a small interval $[-\rho, \rho]$, this means that the hypotheses of the cases are effectively satisfied more often.

3.4. Computational results. We first examine the Jacobian for a network with random initial weights on all layers and give an example. After that we present the singular values of the Jacobian and Hessian (4) for some training problems at later iterations of a Levenberg–Marquardt algorithm.

The singular value decomposition is used to measure ill-conditioning. Although the rank of J is equal to the number of its nonzero singular values, numerically it is rare to compute a singular value exactly equal to zero. To avoid the need for ad hoc numerical determinations of rank, we present all of the singular values for a given problem. Singular values and eigenvalues were computed using Matlab 3.5 running on a Sun Sparcstation-1. Jacobian and Hessian matrices were computed with 48-bit mantissa arithmetic, using a form of automatic differentiation (see [16] for implementation details).

The 2-3-2 network in Fig. 1 has 19 parameters or weights and so the Jacobian has 19 singular values. Our two-dimensional input space is sampled on a uniform mesh in $[0,1]^2$ with 400 training data points in each case presented, and a maximum of 2000 function evaluations is allowed in the Levenberg–Marquardt algorithm unless indicated otherwise. The implementation of the Levenberg–Marquardt algorithm used here was written by Moré and is available in MINPACK. Note that the initial Jacobian is *independent* of the particular problem one is solving, but *dependent* on the network and the sample distribution of the input points, t_i.

Figures 6 and 7 show the singular values of J for different sets of initial weights chosen randomly from the intervals $(-1,1)$ and $(-100,100)$, respectively. Other commonly used intervals differ only by scale from these two. The random seeds were kept fixed and only the weight interval was changed. The condition numbers of the Jacobian are of order 10^7 and 10^{55}, respectively. Hence the condition number of J^TJ, used in the Newton-like methods of Table 2, is close to or smaller than the inverse of the machine epsilon for a 48-bit mantissa computer. As an example of a larger network, Fig. 8 shows the singular values of the initial Jacobian for four cases for a 5-7-2 network with weights chosen randomly in the region $(-1,1)$ and t_i sampled randomly in the cube $[0,1]^5$.

From the graphs we can conclude that the Jacobian becomes more ill conditioned as the norm of the weight vector increases. Considering that the initial conditions of the Jacobian in Figs. 6 and 7 are not unusual (since often the initial weights are chosen randomly from these two intervals), the numerical method starts with an ill-conditioned J. In Fig. 7 the weight vectors have a larger norm on average than in the other figure of the random weights and are representative of the situation during the solution process.

Next we will investigate an extreme example for the primary sources of rank-deficiency in the Jacobian using the notation and classification given in the previous sections. We use a 2-3-2 network because

- the network is simple to visualize (Fig. 1);
- the network does not contain too many superfluous nodes, i.e., ones unnecessary for the solution of the problem and that would create additional rank-deficiency of the Jacobian;
- the network is unlikely to find a good solution since it contains too few nodes. This allows exploration of behavior far from the optimum. However, this does not mean that the Jacobian would necessarily be of full rank for an "ideal" network or a large network.

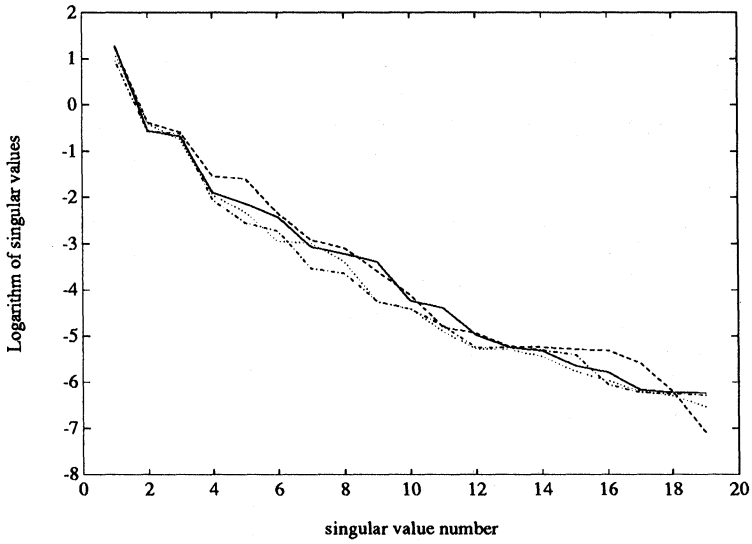

FIG. 6. *Computed singular values of four initial Jacobians for a* 2-3-2 *network with weights chosen randomly from* $(-1, 1)$.

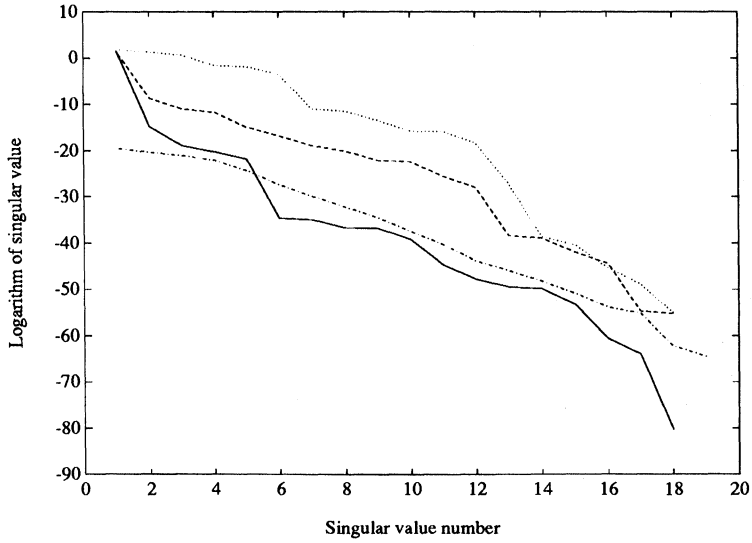

FIG. 7. *Computed singular values of four initial Jacobians for a* 2-3-2 *network with weights chosen randomly from* $(-100, 100)$.

As an example of the different classes of rank-deficiency, we use the initial condition in the (dash-dotted) curve in Fig. 7, where the Levenberg–Marquardt algorithm is allowed to proceed one iteration with a 2-3-2 network. The test figure is a spiral sampled on an equidistant grid as shown in Fig. 9. The value of the function inside the spiral swirls is one, and outside the swirls it is zero.

The results are summarized in Table 3. The cosines of the canonical angles between range spaces are used to measure how closely two blocks J_1 and J_2 of columns of J come to spanning the same space (as is needed for Cases 2 and 3). The method for

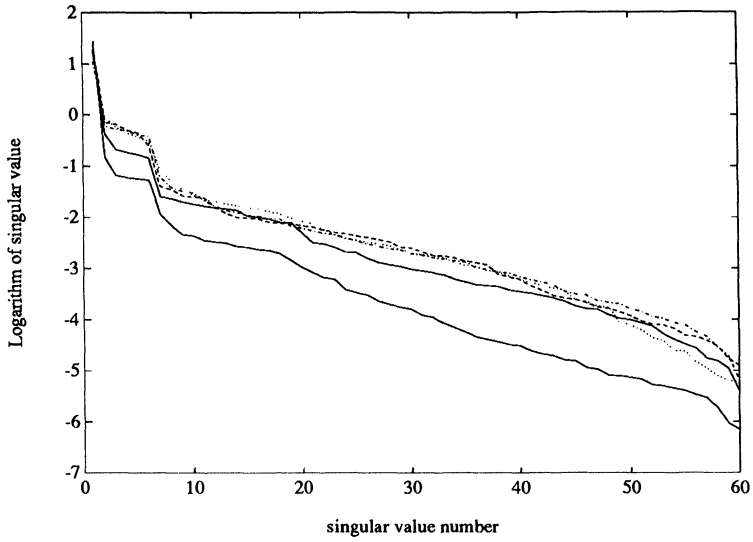

FIG. 8. *Computed singular values of five initial Jacobians for a* 5-7-2 *network.*

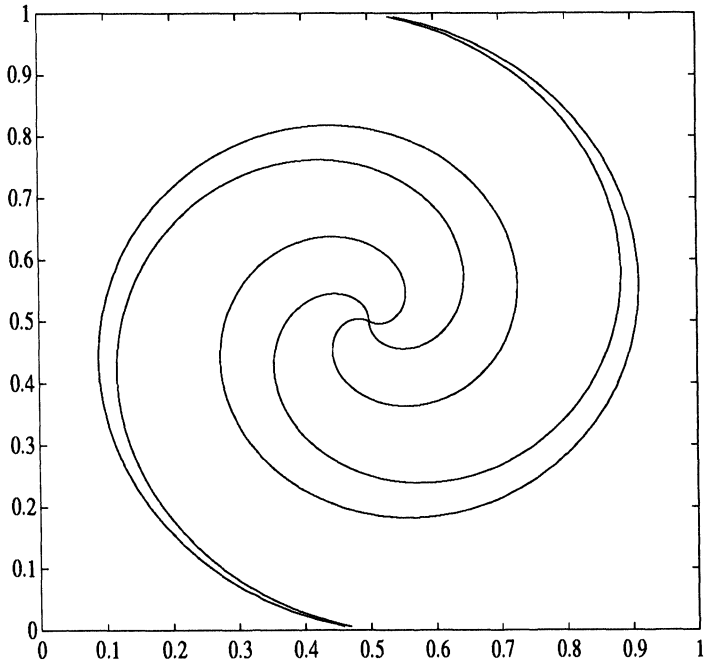

FIG. 9. *The spiral-problem. The function has value* 1 *inside the spiral and* 0 *outside.*

computing such cosines is from [1]. When some cosines are close to 1, then range(J_1) and range(J_2) are close to sharing a subspace spanned by the corresponding canonical vectors. The first column of Table 3 lists the first four dependency cases of §3.3. The second column gives the cosines of the angles between the vectors that determine rank-deficiency (for example, in Case 2 for a 2-3-2 network the determining vectors are $\underline{\sigma'}(Q_1)$ and $\underline{\sigma'}(Q_2)$), the third column specifies the columns of J with a potential

resultant dependency, and the last column gives the cosine(s) of the angle(s) between the affected columns. The claims made in §3.3 are that when the angle between the determining vectors are small, then so are the angles between the affected columns as occurs for this extreme case. The singular values of columns 10–17 and columns 1–9 of the Jacobian are shown together with the singular values of the full Jacobian in Fig. 10.

TABLE 3
Example of rank deficiency.

Dependency case	Determining vectors and cosine of their angle	Affected block columns of J	Canonical cosines for affected cols.
I	e & $\underline{\sigma}(P_1)$ 0.99325	10 & 11 14 & 15	0.99324 0.99326
	e & $\underline{\sigma}(P_2)$ 0.99930	10 & 12 14 & 16	0.99930 0.99931
	e & $\underline{\sigma}(P_3)$ 0.99765	10 & 13 14 & 17	0.99766 0.99765
II	$\underline{\sigma'}(Q_1)$ & $\underline{\sigma'}(Q_2)$ 0.99995	[10,11,12,13] & [14,15,16,17]	1.00000 1.00000 0.99998 0.99993
III	$\underline{\sigma'}(P_1)$ & $\underline{\sigma'}(P_2)$ 0.99733	[1,2,3] & [4,5,6]	1.00000 0.99901 0.99652
	$\underline{\sigma'}(P_1)$ & $\underline{\sigma'}(P_3)$ 0.99639	[1,2,3] & [7,8,9]	1.00000 0.99852 0.99503
	$\underline{\sigma'}(P_2)$ & $\underline{\sigma'}(P_3)$ 0.99861	[4,5,6] & [7,8,9]	1.00000 0.99986 0.99982
IV	$\underline{\sigma}(Q_1)$ & $\underline{\sigma}(Q_2)$ 0.99990	18 & 19	0.99990

Since the conditioning of network Jacobians can be represented succinctly by the singular values and the specific reason for rank-deficiency can be analyzed using the procedure above, we now show only the singular values of a few larger problems.

Figures 11 and 12 show the eigenvalues of J^TJ and the full Hessian for the spiral problem with a 2-14-7 network at some iterations. The initial weights are chosen randomly in the interval $[-1, 1]$. At a solution (Fig. 12) the 40-20-40 rule for a randomly chosen test set of size 1000 gives an error rate of 13.5 percent. This criterion assigns 0 to output values in [0,0.40], and 1 to values in [0.60,1]. Values in [0.40, 0.60] are labeled incorrect. Figure 13 for a 2-16-6 network shows eigenvalues at the solution (the initial weights are chosen from $[-1, 1]$). The Levenberg–Marquardt algorithm appears to be able to obtain an acceptable error norm for this problem for a large range of networks and initial conditions.

3.5. Determining cosines and rank-deficiency. We have monitored the determining and canonical cosines for some of the cases described above in a number of test problem configurations. While rank-deficiency of the Jacobian can result from interactions among columns outside of the groups analyzed separately in Cases 1–5 above, the plots show that the column groups identified above frequently exhibit in-

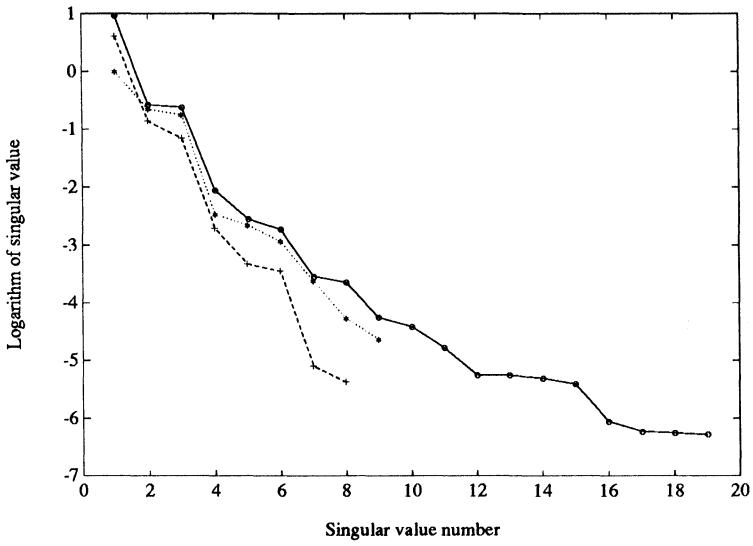

FIG. 10. *Computed singular values of columns* 10–17 (*labeled* + *with dashed line*), 1–9 (*labeled* * *with dotted line*), *and all columns* (*labeled o with solid line*) *of the Jacobian for the example discussed in the text and Table* 3 (*a* 2-3-2 *network used for the spiral problem*).

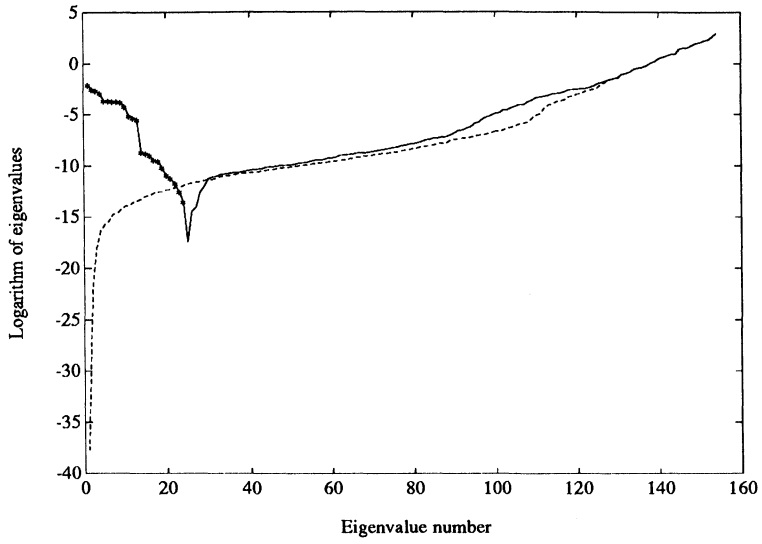

FIG. 11. *The computed eigenvalues of* $J^T J$ (*dashed line*) *and Hessian* (*solid line with* * *indicating negative eigenvalues*) *for the spiral problem with a* 2-14-7 *network at iteration* = 1000 (*the error norm was* $5.2 * 10^{-4}$).

ternal deficiency as predicted. Moreover, large determining cosines do act as sufficient conditions for rank-deficiency.

To demonstrate these facts, we have plotted the canonical and determining cosines for Case 2 in a run of the Levenberg–Marquardt solver. Those cosines as a function of the iteration number for a 2-3-2 network are shown in Fig. 14.

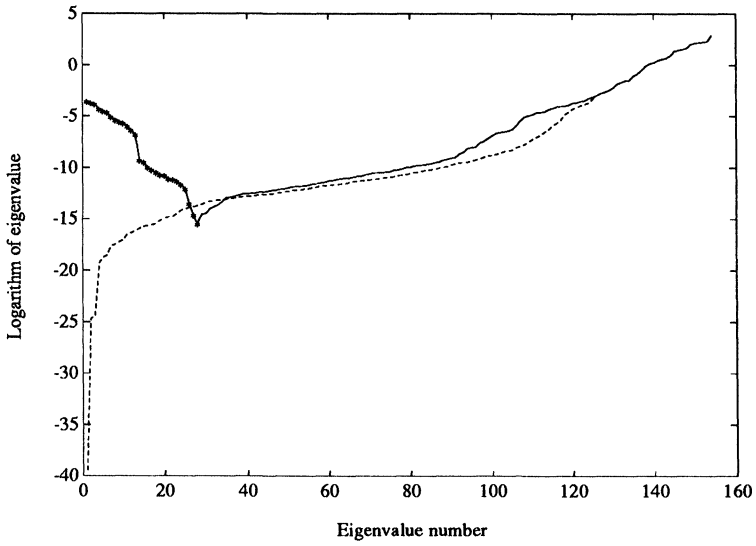

FIG. 12. *The computed eigenvalues of* $J^T J$ *(dashed line) and Hessian (solid line with * indicating negative eigenvalues) for the spiral problem (same as in the previous figure) at the solution (iteration* 1832 *with error norm* $= 4.1 * 10^{-5}$*).*

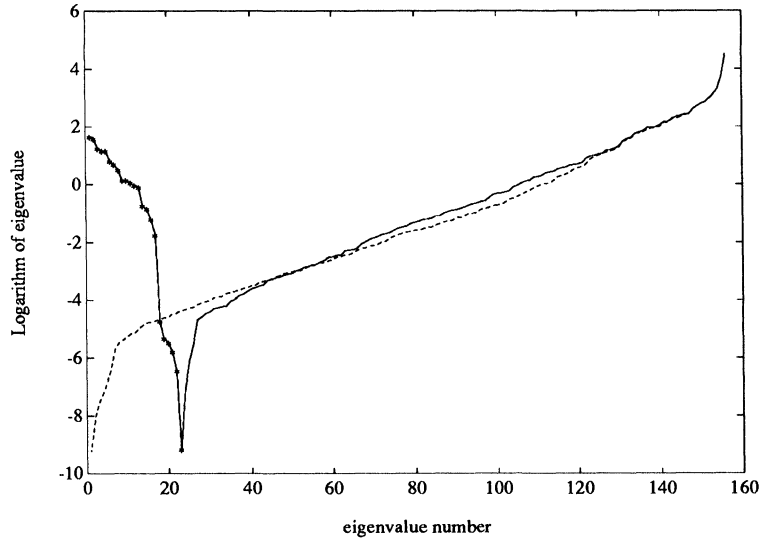

FIG. 13. *Computed eigenvalues of* $J^T J$ *(dashed line) and of the Hessian (solid line, negative eigenvalues are indicated with *) at the last iteration (error norm* $= 3.4$ *for* 900 *training data points and a maximum of* 400 *function evaluations) for the spiral problem with a* 2-16-6 *network. The* 40-20-40 *rule gives an error rate of* 7.1 *percent.*

Additionally, in Figs. 15 and 16 we show the canonical and determining cosines for two 2-8-4 networks, with different initial conditions, at the computed solution for each combination of columns (i.e., six pairs on the second layer). These results suggest that canonical cosines do explain rank-deficiency of the Jacobian in feedforward networks. In Fig. 16, Case 2 does not appear and the rank-deficiency comes from other causes.

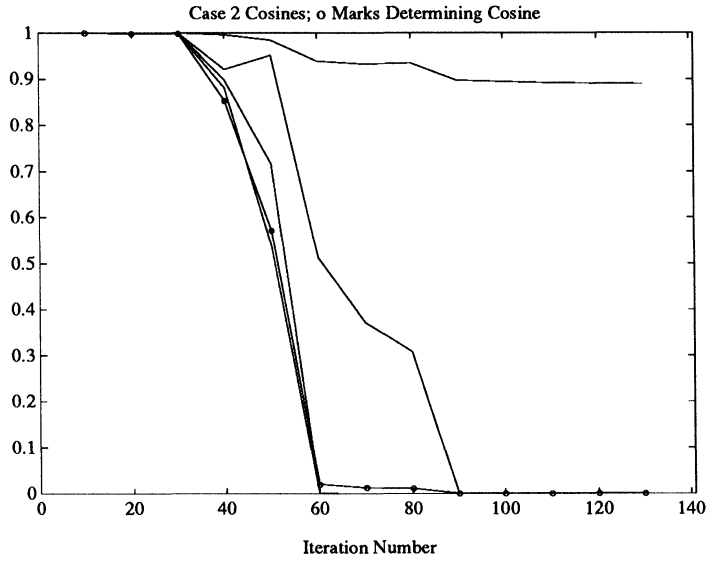

FIG. 14. *The computed cosines for Case 2 for a 2-3-2 network as a function of iteration number. The determining cosines are marked "o."*

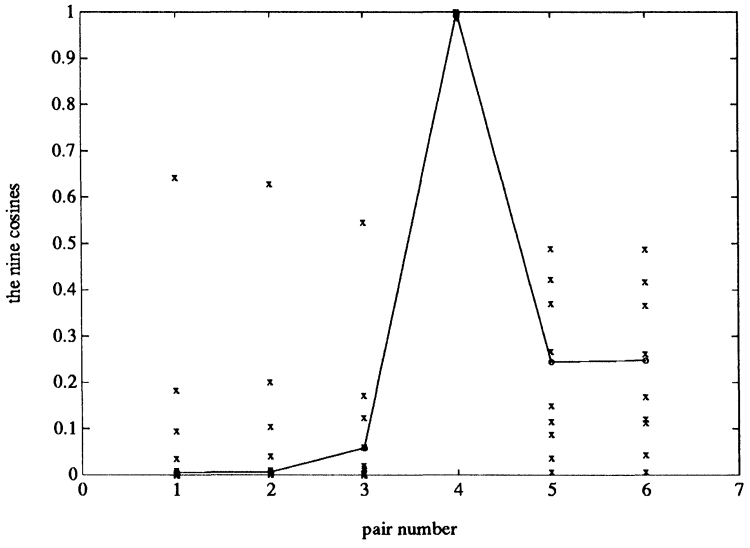

FIG. 15. *The resulting nine cosines from Case 2 for a 2-8-4 network shown versus the six pairs on the second layer. The determining cosines are indicated by "o" and connected with solid lines.*

In Figs. 17 and 18 we show the corresponding eigenvalues of the Hessian and $J^T J$ at the solution for these runs.

3.6. The Jacobian for a 2-3 network, with one hidden layer. This section presents the structure of the Jacobian for a one hidden layer network of size 2-3, that is, with two inputs and three nodes on the hidden layer. These networks have been successfully used in many neural network applications [17] and they can be used also

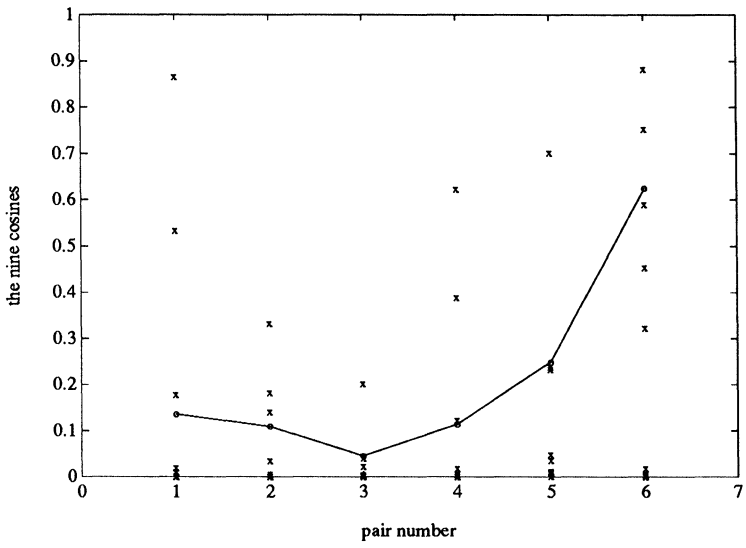

FIG. 16. *The resulting nine cosines from Case* 2 *for a* 2-8-4 *network shown versus the six pairs on the second layer. The determining cosine are indicated by "o" and connected with solid lines.*

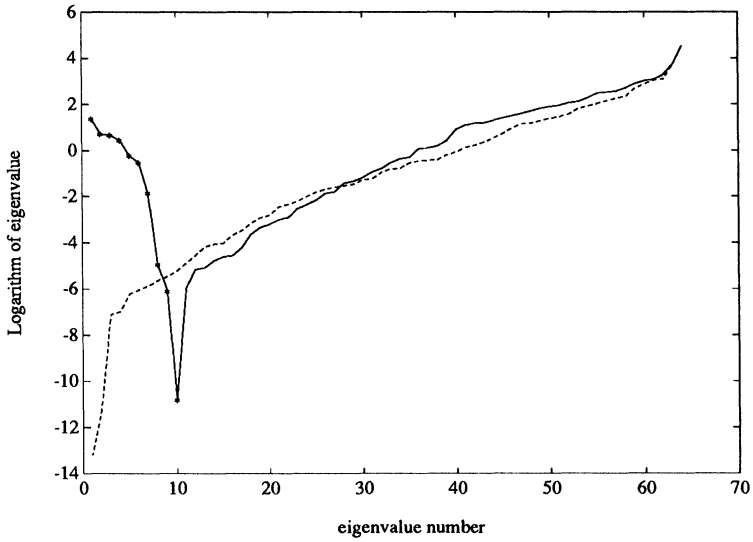

FIG. 17. *The computed eigenvalues for the Hessian (solid line with * indicating negative eigenvalues) and* J^TJ *(dashed line) at the solution for a* 2-8-4 *network (see also Fig.* 15). *The* 40-20-40 *rule gives the error rate* 5.5 *percent with a maximum of* 400 *iterations.*

as universal function approximators [3]. The form of the one hidden layer Jacobian is shown below with a numbering similar to that in Fig. 1.

$$J_{i,1\ldots3} = x_{10}\sigma'(P_1^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T,$$
$$J_{i,4\ldots6} = x_{11}\sigma'(P_2^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T,$$
$$J_{i,7\ldots9} = x_{12}\sigma'(P_3^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T,$$
$$J_{i,10\ldots12} = (\sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T.$$

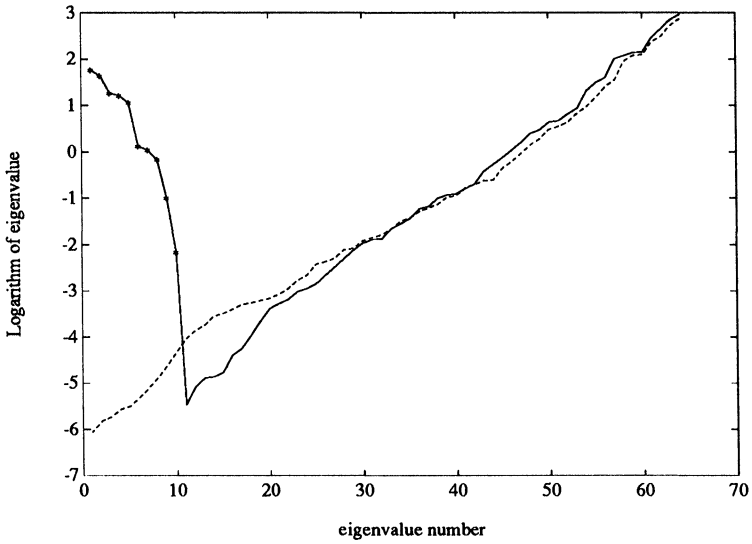

FIG. 18. *The computed eigenvalues for the Hessian (solid line with * indicating negative eigenvalues) and J^TJ (dashed line) at the solution for a* 2-8-4 *network (see also Fig.* 16). *The* 40-20-40 *rule gives the error rate* 15 *percent with a maximum of* 400 *iterations.*

The Jacobian for such a system is also likely to be rank-deficient because situations as in Case 2 appear in the equations. Columns 1–9 of the Jacobian above have the same three terms in common, i.e., the vector $(1, v_1^{(i)}, v_2^{(i)})$, which are multiplied by a term of the form $\sigma'(x)$.

4. Summary and conclusions. In a feedforward neural network the Jacobian is usually ill conditioned. Since many numerical schemes use it as a basis for their search direction, any rank-deficiency in the Jacobian causes the algorithm to obtain only partial information of the possible search directions, and in turn causes long training times. The rank-deficiency for a two hidden layer network often arises from the outputs of the second layer nodes because a sigmoidal applied to these outputs has a limited discrimination capability, especially if the weights for this level become large. Consequently, for a network with more hidden layers the rank-deficiency can only increase, but the rank-deficiency for a one hidden layer network could in principle be less severe.

Our experiments and analyses of partially connected networks and of single hidden layer networks do not indicate a significantly better conditioned Jacobian. Use of partially connected networks may inhibit rank-deficiency somewhat, but cannot guarantee complete avoidance of the intrinsic problems. A situation where some connections are omitted between two consecutive hidden layers appears in the Jacobian as the omission of some terms in the sum in (6). The apparent redundancy of a Jacobian can only be eliminated by a posteriori deletion of nodes and weights in a problem-specific way, so there would be no effect on the training process itself. Again, one could still have rank-deficient Jacobians arising in problems with either zero or nonzero residuals using reduced connectivity networks.

The Jacobian conditioning can possibly be improved by using a different form of the sigmoidal (i.e., a function with a better discrimination capability). A better initial condition might cause the Levenberg–Marquardt algorithm to find a solution with a

weight vector that has a smaller norm, but practice shows that the weight vector at the solution has a large norm. The large norm of the weight vector (cf., (5)–(7)) is one reason that the algorithm causes the $\sigma'(x)$-function to approach its limit zero and therefore causes rank-deficiency in the Jacobian, as explained in §3.3.

In short, formulating feedforward neural network problems as least squares problems can cause undue strain on any numerical scheme which uses Jacobians, and a reformulation of the problem is called for.

Acknowledgments. The authors thank the anonymous referees and especially the Associate Editor, Margaret Wright, for their corrections and helpful suggestions, which have greatly improved this paper.

REFERENCES

[1] A. BJÖRCK AND G. GOLUB, *Numerical methods for computing angles between linear subspaces*, Math. Comput., 27 (1973), pp. 579–594.

[2] M. D. BUHMANN, *Multivariate interpolation in odd dimensional Euclidean spaces using multiquadratics*, Tech. Rep. DAMTP 1988/NA6, Dept. of Applied Mathematics and Theoretical Physics, Univ. of Cambridge, Cambridge, U.K., 1988.

[3] G. CYBENKO, *Approximations by superpositions of a single function*, Math. Control Signals Systems, 2 (1989), pp. 303–314.

[4] J. DENNIS AND R. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[5] C. FRALEY, *Solution of nonlinear least-squares problems*, Ph.D. thesis, Stanford Univ., Stanford, CA, June 1987; Tech. Rep. # STAN-CS-87-1165, Dept. of Computer Science, Stanford University, Stanford, CA.

[6] P. GILL, W. MURRAY, AND M. WRIGHT, *Practical Optimization*, Academic Press, London, New York, 1981.

[7] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, 2nd ed., The John Hopkins University Press, Baltimore, MD, 1989.

[8] J. HERTZ, A. KROGH, AND R. G. PALMER, *Introduction to the Theory of Neural Networks*, Addison-Wesley, Reading, MA, 1991.

[9] D. LUENBERGER, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1973.

[10] J. J. MORÉ, *The Levenberg–Marquardt algorithm: Implementation and theory*, in Numerical Analysis, Lecture Notes in Math., Vol. 630, G. A. Watson, ed., Springer-Verlag, Berlin, New York, 1977, pp. 105–116.

[11] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[12] M. POWELL, *Radial basis functions for multivariable interpolation: A review*, in IMA Conference on Algorithms for the Approximation of Functions and Data, Oxford University Press, London, U.K., 1987.

[13] M. J. D. POWELL, *Restart procedures for the conjugate gradient method*, Math. Programming, 12 (1977), pp. 241–254.

[14] ———, *Approximation theory and methods*, Cambridge University Press, London, U.K., 1981.

[15] D. E. RUMELHART AND J. L. MCCLELLAND, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986.

[16] S. SAARINEN, R. BRAMLEY, AND G. CYBENKO, *Neural networks, backpropagation, and automatic differentiation*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. Corliss, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992, pp. 31–42.

[17] T. SEJNOWSKI AND C. ROSENBERG, *Parallel networks that learn to pronounce English text*, Complex Systems, 1 (1987), pp. 1134–1142.