Learning Algorithms for Neural Networks Based on Quasi-Newton Methods With Self-Scaling

H. S. M. Beigi[1]

C. J. Li

Department of Mechanical Engineering,
Columbia University,
New York, NY 10027

Previous studies have suggested that, for moderate sized neural networks, the use of classical Quasi-Newton methods yields the best convergence properties among all the state-of-the-art [1]. This paper describes a set of even better learning algorithms based on a class of Quasi-Newton optimization techniques called Self-Scaling Variable Metric (SSVM) methods. One of the characteristics of SSVM methods is that they provide a set of search directions which are invariant under the scaling of the objective function. With an XOR benchmark and an encoder benchmark, simulations using the SSVM algorithms for the learning of general feedforward neural networks were carried out to study their performance. Compared to classical Quasi-Newton methods, it is shown that the SSVM method reduces the number of iterations required for convergence by 40 percent to 60 percent that of the classical Quasi-Newton methods which, in general, converge two to three orders of magnitude faster than the steepest descent techniques.

1 Introduction

This paper describes new learning algorithms for feedforward neural networks. In such a network, a neuron sums a number of weighted inputs and a bias and passes the result through a nonlinear activation function. Multilayer neural networks consist of a large number of these neurons. Before a neural network can be used for any purpose, the weights connecting inputs to neurons and bias values should be adjusted so that outputs of the network will match desired patterns for specific sets of inputs. The methods used for adjusting these weights and parameters to provide such a match are usually refered to as learning algorithms.

Rumelhart et al. [2] rederived independently and brought the back-propagation learning algorithm for multilayer feedforward neural nets to the attention of the community in 1986. They used the generalized delta rule to compute the needed gradient for this steepest decent method. However, low rates of convergence were seen in practically every problem. Lippmann [3] states, "One difficulty noted by the backward propagation algorithm is that in many cases the number of presentations of training data required for convergence has been large (more than 100 passes through all the training data)." A few methods [3–4] have been proposed to increase the rate of convergence of learning by making very restrictive assumptions such as linearity. Other more practical methods have recently been proposed for accelerating the convergence of the back-propagation technique (see for example [5–11]).

Reference [7] used a relative entropy as the error measure instead of the quadratic function and the steepest descent is used to find the minimum. Reference [8] formulated the learning problem into solving a set of coupled ordinary differential equations and was able to speed up the learning by one order of magnitude. However, the performance index used by the paper is really a misleading one because it does not include the computation load incurred by the solving of the equations. Reference [9] suggested the use of conjugate gradient method which is a first order descent method with an inferior performance to Quasi-Newton methods. Reference [10] derived a second order method which approximates Newton's method. However, the scheme requires the evaluation of a square matrix with components of second partial derivatives which are not available for a general neural network. Reference [11] evaluated back-propagation, steepest descent with line search, momentum method, and classical Quasi-Newton methods and concluded that the use of Quasi-Newton methods is mandated by their excellent convergence properties.

In general, steepest descent techniques have good performance while away from a local minimum and require a lot of iterations to converge while close to the minimum. On the other hand, Newton's method usually converges fast in the vicinity of the minimum. In addition, Newton's minimization technique handles functions with ill-conditioned Hessian matrices elegantly [12]. It would be desirable to take advantage of the properties of steepest descent when the state is far from the minimum and then to use Newton's method in the vicinity of the minimum.

For using Newton's method, the first gradient and the matrix of second partial derivatives (Hessian) should be evaluated. Due to the absence of a general explicit expression for the Hessian matrix of a feedforward neural network, one would settle for its approximation. One way would be to look at the problem in a manner similar to Rumelhart's technique and to

[1]Presently, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

Contributed by the Dynamic Systems and Control Division for publication in the JOURNAL OF DYNAMIC SYSTEMS, MEASUREMENT, AND CONTROL. Manuscript received by the Dynamic Systems and Control Division July 16, 1990; revised manuscript received June, 1992. Associate Technical Editor: R. Shoureshi.

to use a momentum method which would approximate the diagonal elements of the Hessian matrix and stay ignorant of the off-diagonal elements [13].

The difficulties associated with the use of Newton's method for neural network learning are: 1) the absence of a general explicit expression for the Hessian matrix of a neural network; 2) the inversion of the Hessian; and 3) the optimal switching from steepest descent to Newton's method.

In [1], the authors investigated nine different Quasi-Newton methods and the results were compared to a steepest descent technique with line search. Quasi-Newton methods use an iterative process to approximate the inverse Hessian matrix so that no explicit expression for the second derivative is needed for carrying out a Newton-like search. If one selects the initial estimate to be an identity matrix, it is a steepest descent technique at the beginning and gradually changes into Newton's method as the estimate becomes the inverse of the Hessian. This paper describes a set of even better learning algorithms based on an improved class of Quasi-Newton optimization techniques called Self-Scaling Variable Metric (SSVM) methods. One of the characteristics of SSVM methods is that they provide a set of search directions which are invariant under the scaling of the objective function [14–15]. In practice, it has been observed that the SSVM methods have better convergence proprieties than the classical Quasi-Newton methods.

2 Problem Formulation

In this section, the learning problem will be treated within the context of nonlinear optimization. Section 2.1 formulates an objective function and provides an explicit expression for the gradient of the objective function. Section 2.2 applies the results under Section 2.1 to an example of a three-layer neural network that consists of three nonlinear neurons of sigmoid activation function.

2.1 The Objective Function, State Vector, and Gradient Evaluation. The learning problem of feedforward neural networks will be formulated within the context of nonlinear optimization. Figure 1 is a multilayer feedforward neural network consisting of L layers, made of a number of neurons. The network is fully interconnected from one layer to the next layer and the connections are represented by lines which are characterized by their weights. Based on the weights of all the input connections, each neuron computes a weighted sum of all the inputs and evaluates a nonlinear activation function such as a logistic function using the sum. The result of this function evaluation is the output of the neuron. The objective of a learning algorithm is to find the weights and the bias values, to minimize the discrepancy between the outputs of a neural network at layer L and the desired outputs, corresponding to a set of specific inputs. The set of input-output patterns used for the learning is termed as "exemplars." Let us define:

$l \in [1,L]$	where L is the number of layers in the network
$n_l \in [1,N_l]$	where N_l is the number of neurons in layer l
$p \in [1,P]$	where P is the number of input-output patterns in the training exemplars
ω^l_{nm}	as the weight of the mth input to neuron n in layer l
o^l_{pn}	as the output of neuron n in layer l for input pattern p
t_{pn}	as the desired output of neuron n in layer L for input pattern p
i_{pm}	as the mth input component of the input pattern p to the network
ϕ^l_n	as the bias, or the threshold, or neuron n in layer l

Based on the above definitions, the following vectors are defined. A weight vector of all the weights of a neuron, say, neuron n in layer l:

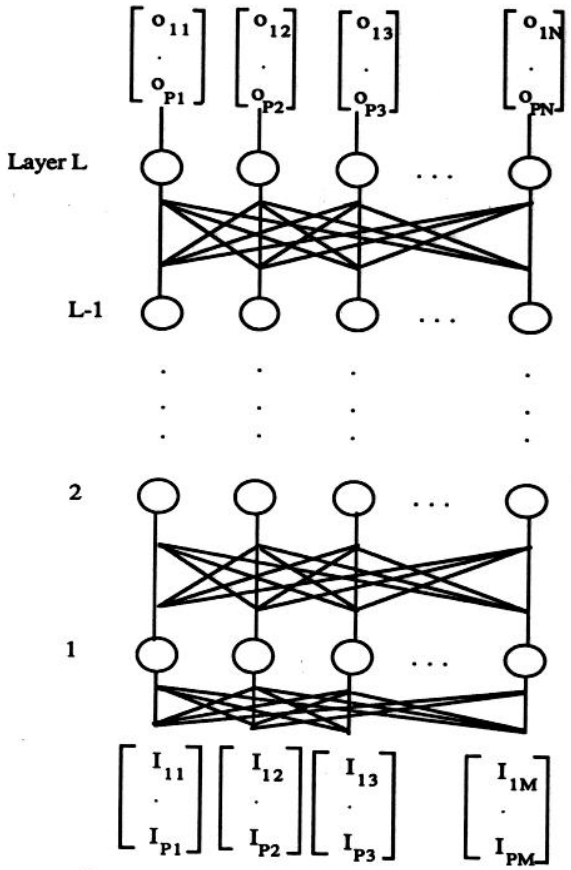

Fig. 1 Multilayer feedforward neural network

$$\omega^{lT}_{n_l} = [\omega^l_{n_l 1}, \omega^l_{n_l 2}, \ldots, \omega^l_{n_l N_{l1}}]$$

A vector formed by concatenating all the weight vectors in layer l:

$$\omega^{lT} = [\omega^{lT}_1, \omega^{lT}_2, \ldots, \omega^{lT}_{N_l}]$$

A vector of all the bias values in layer l:

$$\phi^{lT} = [\phi^l_1, \phi^l_2, \ldots, \phi^l_{N_l}]$$

A state vector, which contains all the state variables to be adjusted in layer l, formed by concatenating ω^{lT} and ϕ^{lT}:

$$\mathbf{x}^{lT} = [\phi^{lT}, \omega^{lT}]$$

Finally, a super state vector, which contains all the state variables to be adjusted in the neural network, formed by concatenating all the state vectors:

$$\mathbf{x}^T = [\mathbf{x}^{1T}, \mathbf{x}^{2T}, \ldots, \mathbf{x}^{LT}]$$

Let's formally define our objective function as the sum of the square of output errors in the output layer (layer L) of a neural network over a set of exemplars. Then,

$$E(\mathbf{x}) = \sum_{p=1}^{P} \sum_{n_L=1}^{N_L} (o^L_{pn_L} - t_{pn_L})^2 \tag{1}$$

Define,

$$E_p = \sum_{n_L=1}^{N_L} (o^L_{pn_L} - t_{pn_L})^2$$

then,

$$E = \sum_{p=1}^{P} E_p \tag{2}$$

Consequently, the gradient vector of the objective function is

$$g = \nabla_x E = \sum_{p=1}^{P} \nabla_x E_p = [\partial E/\partial x_1,\ \partial E/\partial x_2 \ldots\ldots]^T$$

and the Hessian matrix of the objective function is

$$G = \nabla_x^2 E = \sum_{p=1}^{P} \nabla_x^2 E_p \quad \text{(Hessian Matrix)}$$

$$= \begin{bmatrix} \dfrac{\partial^2 E}{\partial x_1 \partial x_1} & \dfrac{\partial^2 E}{\partial x_1 \partial x_2} & \cdots \\ \dfrac{\partial^2 E}{\partial x_2 \partial x_1} & \dfrac{\partial^2 E}{\partial x_2 \partial x_2} & \cdots \\ \ldots\ldots\ldots\ldots\ldots\ldots & & \end{bmatrix}$$

The dimension of the Hessian is the square of the dimension of the super-state vector x.

To find the gradient vector of the objective function, one has to find partial derivatives of the objective function with respect to any component, x_j, of the state vector $\mathbf{x}$.

$$\frac{\partial E}{\partial x_j} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial x_j} \tag{3}$$

and by chain rule,

$$\frac{\partial E_p}{\partial x_j} = 2 \sum_{n_L=1}^{N_L} (o_{pn_L}^L - t_{pn_L}) \frac{\partial o_{pn_L}^L}{\partial x_j} \tag{4}$$

where, if x_j is the bias

$$\frac{\partial o_{pn_L}^L}{\partial \phi_{n_l}^L} = \frac{\partial o_{pn_L}^L}{\partial o_{pn_l}^l} \frac{\partial o_{pn_l}^l}{\partial \phi_{n_l}^l} \tag{5a}$$

or if x_j is a weight

$$\frac{\partial o_{pn_L}^L}{\partial \omega_{n_l n_{l1}}^l} = \frac{\partial o_{pn_L}^L}{\partial o_{pn_l}^l} \frac{\partial o_{pn_l}^l}{\partial \omega_{n_l n_{l1}}^l} \tag{5b}$$

and,

$$\frac{\partial o_{pn_L}^L}{\partial o_{pn_l}^l} = \prod_{i=l+1}^{L} \frac{\partial o_{pn_{(L+l-i+1)}}^{(L+l-i+1)}}{\partial o_{pn_{(L+l-i)}}^{(L+l-i)}} \quad \text{(using the index notation)} \tag{6}$$

In Eq. (6) the indicial notation has been employed, i.e.,

$$\frac{\partial o_{pn_{l+1}}^{l+1}}{\partial o_{pn_l}^l} \frac{\partial o_{pn_l}^l}{\partial o_{pn_{l-1}}^{l-1}} \sum_{n_l=1}^{N_l} \frac{\partial o_{pn_{l+1}}^{l+1}}{\partial o_{pn_l}^l} \frac{\partial o_{pn_l}^l}{\partial o_{pn_{l-1}}^{l-1}} \quad \text{(traditional notation)}$$

In our implementation of neural networks, the following logistic function is used for the activation function

$$o_{pn_l}^l = \frac{1}{1 + e^{-s_{pn_l}^l}} \tag{7}$$

where,

$$s_{pn_l}^l = \sum_{n_{l-1}=1}^{N_{l-1}} (o_{pn_{l-1}}^{l-1} \omega_{n_l n_{l-1}}^l) + \phi_{n_l}^l$$

Then

$$\frac{\partial o_{pn_l}^l}{\partial \phi_{n_l}^l} = d_{pn_l}^l \tag{8}$$

$$\frac{\partial o_{pn_l}^l}{\partial \omega_{n_l n_{l-1}}^l} = d_{pn_l}^l o_{pn_{l-1}}^{l-1} \tag{9}$$

and,

$$\frac{\partial o_{pn_l}^l}{\partial o_{pn_{l-1}}^{l-1}} = d_{pn_l}^l \omega_{n_l n_{l-1}}^l \tag{10}$$

where,

$$d_{pn_l}^l = \frac{e^{-s_{pn_l}^l}}{(1 + e^{-s_{pn_l}^l})^2} \tag{11}$$

However, from Eq. (7),

$$e^{-s_{pn_l}^l} = \frac{1}{o_{pn_l}^l} - 1 \tag{12}$$

Rewrite Eq. (11) using (12),

$$d_{pn_l}^l = o_{pn_l}^l (1 - o_{pn_l}^l) \tag{13}$$

Substitute (13) into (8), (9), and (10),

$$\frac{\partial o_{pn_l}^l}{\partial \phi_{n_l}^l} = o_{pn_l}^l (1 - o_{pn_l}^l) \tag{14}$$

$$\frac{\partial o_{pn_l}^l}{\partial \omega_{n_l n_{l-1}}^l} = o_{pn_{l-1}}^{l-1} o_{pn_l}^l (1 - o_{pn_l}^l) \tag{15}$$

and,

$$\frac{\partial o_{pn_l}^l}{\partial o_{pn_{l-1}}^{l-1}} = \omega_{n_l n_{l-1}}^l o_{pn_l}^l (1 - o_{pn_l}^l) \tag{16}$$

For this logistic activation function, one uses Eqs. (14–16) to evaluate Eq. (5) and then Eq. (4). Finally, components of the gradient vector are evaluated using Eq. (3). In the following section, an example of deriving an explicit expression for the gradient vector will be given.

The purpose of the above derivations is to derive the gradient vector for the least square objective function (1). Back-propagation algorithm does not provide components of the gradient vector explicitly and does not have the gradient vector in a vector form which is needed for the Quasi-Newton method. Additionally, the back-propagation method is really just an implementation of the steepest descent method. A derivation based on the genuine mathematic notations such as the chain rule and the gradient vector can provide a more general perspective.

2.2 Example: The Exclusive-OR (XOR) Problem. Take the example of a network of three neurons (Fig. 2) which is

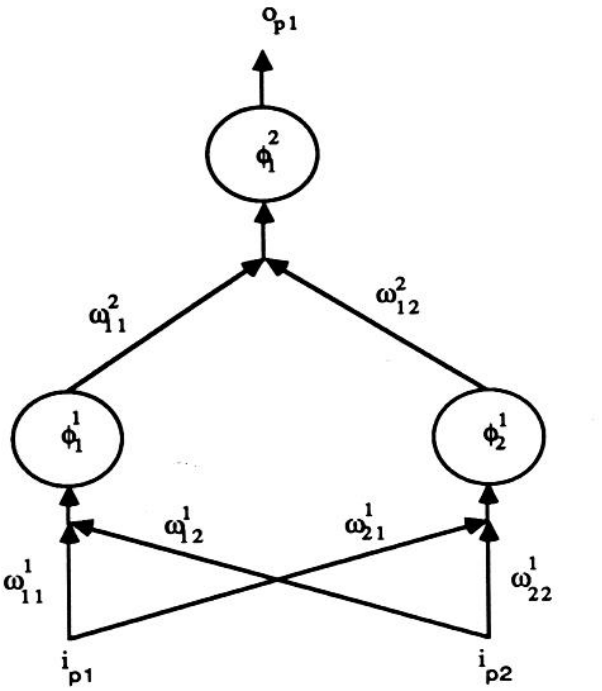

Fig. 2 The exclusive-or network

Table 1 Desired input-output patterns for the XOR benchmark

Input patterns		Output patterns
0	0	0
1	0	1
0	1	1
1	1	0

employed to simulate the exclusive-OR (XOR) logic pattern (Table 1).

The objective function of minimization will then be,

$$E=\sum_{p=1}^{4}(o_{p1}^2-t_{p1})^2$$

and the super state vector is defined by the following sequence of definitions,

$$\phi^{1T}=[\phi_1^1,\phi_2^1]$$

$$\phi^2=[\phi_1^2]$$

$$\omega^{1T}=[\omega_{11}^1,\omega_{12}^1,\omega_{21}^1,\omega_{22}^1]$$

$$\omega^{2T}=[\omega_{11}^2,\omega_{12}^2]$$

$$\mathbf{x}^T=[\phi_1^1,\phi_2^1,\omega_{11}^1,\omega_{12}^1,\omega_{21}^1,\omega_{22}^1,\phi_1^2,\omega_{11}^2,\omega_{12}^2]$$

From Eq. (4),

$$\frac{\partial E_p}{\partial x_j}=2(o_{p1}^2-t_{p1})\frac{\partial o_{p1}^2}{\partial x_j}$$

and from Eqs. (14–16),

$$\frac{\partial o_{p1}^2}{\partial \phi_1^2}=o_{p1}^2(1-o_{p1}^2)$$

$$\frac{\partial o_{p1}^2}{\partial \omega_{11}^2}=o_{p1}^1 o_{p1}^2(1-o_{p1}^2)$$

$$\frac{\partial o_{p1}^2}{\partial \omega_{12}^2}=o_{p2}^1 o_{p1}^2(1-o_{p1}^2)$$

$$\frac{\partial o_{p1}^2}{\partial \phi_1^1}=\frac{\partial o_{p1}^2}{\partial o_{p1}^1}\frac{\partial o_{p1}^1}{\partial \phi_1^1}=\omega_{11}^2 o_{p1}^2(1-o_{p1}^2)o_{p1}^1(1-o_{p1}^1)$$

$$\frac{\partial o_{p1}^2}{\partial \phi_2^1}=\frac{\partial o_{p1}^2}{\partial o_{p2}^1}\frac{\partial o_{p2}^1}{\partial \phi_2^1}=\omega_{12}^2 o_{p1}^2(1-o_{p1}^2)o_{p2}^1(1-o_{p2}^1)$$

$$\frac{\partial o_{p1}^2}{\partial \omega_{11}^1}=\frac{\partial o_{p1}^2}{\partial o_{p1}^1}\frac{\partial o_{p1}^1}{\partial \omega_{11}^1}=\omega_{11}^2 o_{p1}^2(1-o_{p1}^2)i_{p1}o_{p1}^1(1-o_{p1}^1)$$

$$\frac{\partial o_{p1}^2}{\partial \omega_{12}^1}=\frac{\partial o_{p1}^2}{\partial o_{p1}^1}\frac{\partial o_{p1}^1}{\partial \omega_{12}^1}=\omega_{11}^2 o_{p1}^2(1-o_{p1}^2)i_{p2}o_{p1}^1(1-o_{p1}^1)$$

$$\frac{\partial o_{p1}^2}{\partial \omega_{21}^1}=\frac{\partial o_{p1}^2}{\partial o_{p2}^1}\frac{\partial o_{p2}^1}{\partial \omega_{21}^1}=\omega_{12}^2 o_{p1}^2(1-o_{p1}^2)i_{p1}o_{p2}^1(1-o_{p2}^1)$$

$$\frac{\partial o_{p1}^2}{\partial \omega_{22}^1}=\frac{\partial o_{p1}^2}{\partial o_{p2}^1}\frac{\partial o_{p2}^1}{\partial \omega_{22}^1}=\omega_{12}^2 o_{p1}^2(1-o_{p1}^2)i_{p2}o_{p2}^1(1-o_{p2}^1)$$

Using the above equations, the elements of the gradient vector for an exemplar, $\nabla_x E_p$, can be evaluated and then the elements of the gradient vector, $\nabla_x E$, are found using Eq. (3).

3 Variable Metric (Quasi-Newton) Minimization Techniques

First, Quasi-Newton minimization techniques will be briefly reviewed. Let x^* denote the minimum of the objective function E. Further assume that the current state is x_k and define Δx_k to be the difference between the states at x^* and x_k, namely,

$$x^*=x_k+\Delta x_k \tag{17}$$

Write the Taylor series expansion of $E(x^*)$ about x_k assuming that every update of the state vector should drive the state vector to the optimal value x^*,

$$E(x^*)=E(x_k)+\nabla_x^T E|_{x_k}\Delta x_k+\frac{1}{2}\Delta x_k^T\nabla_x^2 E|_{x_k}\Delta x_k+O(\Delta x_k^3) \tag{18}$$

By using the previously defined symbols for the first and second gradients of E (g and G), with a subscript k to denote their evaluation at x_k, we may write (18) as,

$$E(x^*)=E(x_k)+g_k^T\Delta x_k+\frac{1}{2}\Delta x_k^T G_k\Delta x_k+O(\Delta x_k^3) \tag{19}$$

If we disregard the higher than second order terms in (19) and thus approximate E with a quadratic function in the vicinity of x_k and x^*, then the quadratic approximation of (19) may be written as follows:

$$E(x^*)\approx E(x_k)+g_k^T\Delta x_k+\frac{1}{2}\Delta x_k^T G_k\Delta x_k \tag{20}$$

Note that for a minimum of $E(x^*)$, a necessary condition is that $\nabla_{x^*}E$ be zero. However, keeping the current state x_k constant and then taking the gradients of both sides of (20), since $x^*=x_k+\Delta x_k$,

$$\nabla_{x^*}E\approx g_k+G_k\Delta x_k \tag{21}$$

Setting the gradient of E at x^* to zero gives,

$$g_k+G_k\Delta x_k\approx 0 \tag{22}$$

or,

$$\Delta x_k\approx -G_k^{-1}g_k \tag{23}$$

Decompose Δx_k into a direction s_k and a magnitude λ_k,

$$\Delta x_k=\lambda_k s_k \tag{24}$$

where

$$s_k=-\frac{G_k^{-1}g_k}{\|G_k^{-1}g_k\|}$$

Since E is not a quadratic function in x_k, the step size λ_k may not be the optimal one. Therefore, a line search method is usually used to find λ_k^* along direction s_k to minimize the function.

$$x_{k+1}=x_k+\lambda_k^* s_k \tag{25}$$

This method provides quadratic convergence and is very efficient in the vicinity of the minima. However, there are various problems associated with this approach. The first problem is that in order for E to always descend in value, the matrix G^{-1} should be positive definite. Since E is generally not truly quadratic, G^{-1} could become indefinite or even negative definite. There are many techniques developed to maintain a positive definite approximation of the inverse Hessian matrix (G^{-1}) such that the quadratic information in the Hessian matrix will be used. Among these methods for keeping a positive definite approximation of the inverse Hessian matrix are Greenstadt's method [16], Marquardt [17], Levenberg [18], and the alternative of Goldfeld et al. [19].

The second problem is that for networks with a small number of neurons it might be feasible to find an explicit expression of the Hessian matrix. However, for larger networks it will become a very difficult task.

Additionally, it is not practical to take the inverse of a large Hessian matrix even if the Hessian is available. The time involved in taking this inverse in most cases will be more costly than in taking more steps for convergence using a simpler method such as the steepest descent method.

Problem one can be solved by the noted methods above. However, problems two and three make using Newton's method quite impractical. These limitations are reasons for looking at the following alternatives which in turn will solve the discussed problems and still keep a superlinear rate of convergence.

From Eq. (25), we can write the following generalized recursive algorithm to update the state vector such that a minimum E will be approached:

$$x_{k+1}=x_k-\lambda_k^* H_k \nabla_k E(k) \tag{26}$$

where λ_k^* is a weighting factor, and H_k is a square symmetric matrix. Depending on the matrix H_k that is used in Eq. (26), different optimization algorithms will be provided. Therefore, H_k multiplied by the gradient of E will provide a direction and λ_k^* is the optimal step in that descent direction as provided by some line search method. If H_k in Eq. (26) is made equivalent to the identity (I) matrix, then the method reduces to the steepest descent technique which provides linear convergence. Making H_k equivalent to the inverse of the Hessian matrix (G^{-1}) as previously defined, the method will reduce to the Newton minimization technique which provides quadratic convergence in the vicinity of the local minima.

Quasi-Newton methods will start with an approximation to the inverse-Hessian matrix such as the identity matrix. Variable metric updates for H_k are then used, leading to different types of Quasi-Newton methods. In these methods, a rank-one or rank-two update is provided for H_k, denoted by ΔH_k, which will lead to the convergence of H to G^{-1} for quadratic functions. Updates to matrix H_k are done recursively in different directions of the inverse-Hessian space, based on the information obtained from the function behavior in those directions. Depending on whether these updates are done in one or two directions at a time, rank-one or rank-two methods are generated. Quasi-Newton methods in general try to keep the hereditary relation Eq. (27*a*) satisfied.

$$H\Delta g_k=\Delta x_k \tag{27a}$$

where $\Delta g_k=g_{k+1}-g_k$. Condition (27*a*) is automatically satisfied for a quadratic function if H is the exact inverse Hessian. However, since in Quasi-Newton methods, the inverse Hessian is supposed to be an approximation, and due to causality in updating H, instead of $H_k\Delta g_k=\Delta x_k$, the methods try to keep the following relation satisfied at each step k,

$$H_{k+1}\Delta g_k=\Delta x_k \tag{27b}$$

This relationship is referred to as the Quasi-Newton condition and it means that the inverse Hessian matrix should be updated such that relation Eq. (27*b*) is satisfied.

Reference [1] gives a thorough review of some classical rank-one and rank-two updates used by Quasi-Newton methods and their performance for neural network learning. Here, a mere listing of the equations for the Pearson II and BFGS updates is given by Eqs. (28) and (29), respectively. Pearson II and BFGS were found to yield the best results among all the classic Quasi-Newton methods.

Pearson's No. 2 update is given by the following [20]:

$$\Delta H_k=\frac{(\Delta x_k-H_k\Delta g_k)\Delta x_k^T}{\Delta x_k^T\Delta g_k} \tag{28}$$

It should be noted that Pearson's methods do not guarantee retainment of positive definiteness of the inverse Hessian Matrix. Therefore, it is a good idea to reset the inverse Hessian approximation to Identity every N iterations. Methods of this nature are called cyclic.

In 1970, Broyden [21], Fletcher [22], Goldfarb [23], and Shanno [24] suggested the BFGS update (29) which is dual with the DFP method [25].

$$\Delta H_k=\left(1+\frac{\Delta^T g_k H_k\Delta g_k}{\Delta^T x_k\Delta g_k}\right)\frac{\Delta x_k\Delta x_k^T}{\Delta^T x_k\Delta g_k}-\frac{\Delta x_k\Delta^T g_k H_k+H_k\Delta g_k\Delta x_k^T}{\Delta^T x_k\Delta g_k} \tag{29}$$

The BFGS update has all the qualities of the DFP method plus the fact that it has been noted to work exceptionally well with inexact line searches and a global convergence proof exists [26] for the BFGS. No such proof has yet been done for the convergence of DFP.

Self-Scaling Variable Metric (SSVM) Algorithms. One characteristic of classic Quasi-Newton methods is that the search directions provided by them change if the objective function is scaled. This is obviously undesirable. Self-Scaling Variable Metric (SSVM) methods were developed to provide directions that are invariant under scaling of the objective function. Experience shows that the SSVM methods, in general, are especially advantages for large scale problems. This makes them ideal candidates for neural network learning.

Equation (30) is a Self-Scaling Variable Metric (SSVM) update, for the approximation to the inverse Hessian matrix [14].

$$H_{k+1}=\mu_k\left(H_k-\frac{H_k\Delta g_k\Delta g_k^T H_k}{\Delta g_k^T H_k\Delta g_k}+\theta_k v_k v_k^T\right)+\frac{\Delta x_k\Delta x_k^T}{\Delta x_k^T\Delta g_k} \tag{30}$$

where,

$$v_k=(\Delta g_k^T H_k\Delta g_k)^{1/2}\left(\frac{\Delta x_k}{\Delta x_k^T\Delta g_k}-\frac{H_k\Delta g_k}{\Delta g_k^T H_k\Delta g_k}\right)$$

This SSVM algorithm maintains positive definiteness of the approximation to the inverse Hessian matrix, provided $\Delta x_k^T\Delta g_k>0$ for all k. This condition can be imposed by using a line search method which satisfies the following relation,

$$g_{k+1}^T\Delta x_k\geq\sigma g_k^T\Delta x_k \tag{31}$$

where $\sigma\in[\tau,1]$ and $\tau\in[0,1/2]$ are parameters of the line search termination [25]. Eventually, this means that the curvature estimate should be positive where the updating is done. The same argument is true for lots of other variable metric methods such as the DFP and BFGS methods [1]. For a general nonlinear function, the SSVM algorithm provides a set of search directions which are invariant under scaling of the objective function. Also, for a quadratic objective function, the algorithm has the property that it monotonically reduces the condition number of the inverse Hessian approximate.

Equation (30) leaves a lot of freedom in choosing the parameters μ_k and θ_k. Oren suggested in [14] that μ_k and θ_k be picked in the following manner,

$$\mu_k=\phi_k\frac{g_k^T\Delta x_k}{g_k^T H_k\Delta g_k}+(1-\phi_k)\frac{\Delta x_k^T\Delta g_k}{\Delta g_k^T H_k\Delta g_k}$$

and ϕ_k, $\theta_k\in[0,1]$. This choice will provide a set of μ_k such that,

$$\frac{\Delta x_k^T\Delta g_k}{\Delta g_k^T H_k\Delta g_k}\leq\mu_k\leq\frac{\Delta x_k^T H_k^{-1}\Delta x_k}{\Delta x_k^T\Delta g_k}$$

Another approach for picking the aforementioned parameters, based mainly on heuristics, is to keep μ_k as close as possible to unity and to pick θ_k such as to offset an estimated bias in $\det(H_kG)$ relative to unity [15]. A third approach for picking these parameters is to minimize the condition number of the inverse Hessian matrix which will provide better number stability of the updating algorithm. Reference [15] gives four sets of switching rules for picking these parameters.

4 Simulations and Results

Computer simulations using two benchmarks, the classical XOR and an encoder, have been carried out to study the performance of the SSVM methods. The input and desired output patterns of these two benchmarks are given in Tables 1 and 2, respectively. The neural network used for the XOR problem is shown in Fig. 2. A similar architecture was used for the encoding problem with four hidden neurons and four inputs. Three measures of the performance of the algorithms are the number of times the input patterns had to be presented, the number of updates which were made to the initial guess for the inverse Hessian matrix to achieve convergence, and the number of floating point operations required to achieve con-

Table 2 Desired input-output pattern for the encoder benchmark

I1	I2	I3	I4	Desired output
1	1	1	1	0
1	0	1	1	1
1	0	1	0	0
0	0	1	0	1
0	0	0	0	0

Table 3 Simulation results

		SD	BFGS	Prsn	O(1,.1)*	O(1,.2)	O(1,.3)	O(1,.5)	O(.5,.5)
XOR	*E*	2.8e-5	4e-16	6e-7	9e-9	0	0	----	8e-7
	P	8553	15	36	12	8	11	----	13
	G	8465	6	11	4	3	4	----	5
	F	8e6	2.5e4	2.2e4	1.1e4	8.5e3	1.1e4	----	1.4e4
Encoder	*E*	8e-5	1e-17	2e-7	7e-6	8e-6	3.7e-6	2e-11	1-e8
	P	1352	36	54	30	42	33	20	28
	G	1194	11	16	9	13	10	6	9
	F	3.8e6	5.1e5	1.4e5	1.3e5	1.9e5	1.4e5	8.7e4	1.3e5

*$O(\phi,\theta)$'s denote the two parameters of Eq. (30).
"----" indicates that the algorithm failed to converge in the allotted number of iterations.

vergence. In simulation of all the above algorithms, an inexact line search was used.

In picking the parameters μ_k and θ_k, a trial and error method was used. The switching method of [14] did not perform well compared to prechosen constant parameters. This result agrees with experiments done by Oren [14]. Also, the switching methods of [15] which were demonstrated to have done well were not used here due to their impracticality for neural networks. In these switching methods, an update of the Hessian matrix should be kept as well as the inverse Hessian. This would require a lot of memory when a large network is involved.

Results of the computer simulation of the SSVM algorithms are given in Table 3 together with results obtained from steepest descent and the best two classical Quasi-Newton methods, BFGS and Pearson. In Table 3, E is the final value of the objective function, P is the number of presentations of all input patterns, G is the number of updates made to the initial inverse Hessian estimate, and F is the number of floating point operations (FPO) required to reach convergence. Termination of the minimization took place when $E<10^{-5}$ or when the magnitude of change in objective function in two consecutive iterations, $|E_{k+1}-E_k|$, is less than 10^{-6}.

5 Conclusion

Looking at the state of the arts in neural network learning, a need was noted for faster learning algorithms. Steepest descent techniques are known to perform well, away from the minima, and Newton's method works well, in the vicinity of the minima. Therefore, for a fast learning algorithm, one should take advantage of the best of the two methods. The evaluation and inversion of the Hessian matrix posed a difficult and time-consuming problem for Newton's method. Quasi-Newton methods start off with an estimate of the inverse Hessian matrix equal to the identity matrix. This matrix provides the steepest descent direction. As recursions are done, the Quasi-Newton methods provide an update to the inverse Hessian matrix. This will, in the limit, provide an optimal direction of descent based on the momentum of the inverse Hessian matrix.

Previous studies have suggested that, for moderate sized neural networks, the use of classical Quasi-Newton methods yields the best convergence properties among all the state-of-the-art methods [1]. This paper describes a set of even better learning algorithms based on a class of Quasi-Newton optimization techniques called Self-Scaling Variable Metric (SSVM) methods. One of the characteristics of SSVM methods is that they provide a set of search directions which are invariant under the scaling of the objective function. Compared to classical Quasi-Newton methods, the SSVM method positively reduces the number of floating point operations required for convergence. The superiority of the SSVM method is equally obvious with both benchmarks. To reach the convergence, the SSVM method needs only about 40 percent and 60 percent of FPO's required by the best classical Quasi-Newton method, Pearson II, which needs 0.3 percent and 3.7 percent of FPO's required by the steepest descent in XOR and Encoder problems respectively.

References

1 Beigi, H. S. M., and Li, C. J., "A New Set of Learning Algorithms for Neural Networks," presented at the ISMM International Symposium, Computer Applications in Design, Simulation and Analysis, March 1990, New Orleans, LA.

2 Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, 1986, MIT Press, pp. 675–695.

3 Lippmann, R. P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April 1987, pp. 4–22.

4 Parker, D. B., "A Comparison of Algorithms for Neuron-Like Cells," *Neural Networks for Computing, AIP Conference Proceeding 151*, Snowbird, Utah, 1986, pp. 327–332.

5 Fahlman, S. E., "Faster-Learning Variations on Back-Propagation: *An Empirical Study*,"1988 Connectionist Models Summer School, 1988, pp. 38–51.

6 Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., and Alkon, D. L., "Accelerating the Convergence of the Back-Propagation Method," *Biological Cybernetics*, Vol. 59, 1988, pp. 257–263.

7 Solla, S. A., Levin, E., and Fleisher, M., "Accelerated Learning in Layered Neural Networks," *Complex Systems*, Vol. 2, 1988, pp. 625–640.

8 Owens, A. J., and Filkin, D. L., "Efficient Training of the Back-Propagation Network by Solving a System of Stiff Ordinary Differential Equations," *IEEE/INNS International Conference on Neural Networks*, June 19–22, 1989, Washington, D.C., Vol. 2, pp. 381–386.

9 Makram-Ebeid, S., Sirat, J.-A., and Viala, J.-R., "A Rationalized Error Back-Propagation Learning Algorithm," *Proc. of the 1989 International Joint Conference on Neural Networks*, Vol. 2, Washington, D.C., July 1989, pp. 373–380.

10 Parker, D. B., "Optimal Algorithms for Adaptive Networks: Second Order Back-Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning," *Proceedings of the 1987 IEEE International Conference on Neural Networks*, Vol. 2, pp. 593–600.

11 Watrous, R. L., "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," *Proceedings of the 1987 IEEE International Conference on Neural Networks*, Vol. 2, pp. 619–627.

12 Himmelblau, D. M., *Applied Nonlinear Programming*, McGraw-Hill, New York, 1972.

13 Becker, S., and Le Cunn, Y., "Improving the Convergence of Back-Propagation Learning With Second Order Methods," *1988 Connectionist Models Summer School*, 1988, pp. 29–37.

14 Oren, S. S., "On the Selection of Parameters in Self-Scaling Variable Metric Algorithms," *Mathematical Programming*, Vol. 7, 1974, pp. 351–367.

15 Oren, S. S., and Spedicato, E., "Optimal Conditioning of Self-Scaling Variable Metric Algorithms," *Mathematical Programming*, Vol. 10, 1976, pp. 70–90.

16 Greenstadt, J., "On the Relative Effectiveness of Gradient Methods," *Mathematics of Computation*, Vol. 21, 1967, pp. 360–367.

17 Marquardt, D. W., "An Algorithm of Least Squares Estimation of Nonlinear Parameters," *SIAM Journal*, Vol. 11, 1963, p. 431.

18 Levenberg, K. L., *Quart. Appl. Math.*, Vol. 2, 1944, p. 164.

19 Goldfeld, S. M., Quandt, R. E., and Trotter, H. F., "Maximization by Quadratic Hill Climbing," *Econometrica*, Vol. 34, 1966, p. 541.

20 Pearson, J. D., "Variable Metric Methods of Minimization," *Computer Journal*, Vol. 12, 1969, pp. 171–178.

21 Broyden, C. G., "The Convergence of a Class of Double Rank Minimization Algorithms," Parts I and II, *J. Inst. Maths. Applns.*, Vol. 6, 1970, pp. 76–79, 222–231.

22 Fletcher, R., "A New Approach to Variable Metric Algorithms," *Computer Journal*, Vol. 8, 1970, pp. 317–322.

23 Goldfarb, D., "A Family of Variable Metric Methods Derived by Variational Means," *Mathematics of Computation*, Vol. 24, 1970, pp. 23–26.

24 Shanno, D. F., "Conditioning of Quasi-Newton Method for Function Minimization," *Mathematics of Computation*, Vol. 24, 1970, pp. 647–656.

25 Fletcher, R., *Practical Methods of Optimization*, Wiley, New York, 1987, pp. 54–56.

26 Huang, H. Y., "Unified Approach to Quadratically Convergent Algorithms for Function Minimization," *Journal of Optimization Theory and Applications*, Vol. 5, No. 6, 1970, pp. 405–422.