PERGAMON

Neural Networks 11 (1998) 1607–1622

Neural Networks

Contributed article

A second-order learning algorithm for multilayer networks based on block Hessian matrix

Yi-Jen Wang*, Chin-Teng Lin

Department of Electrical and Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, ROC

Received 18 November 1996; revised 19 June 1998; accepted 19 June 1998

Abstract

This article proposes a new second-order learning algorithm for training the multilayer perceptron (MLP) networks. The proposed algorithm is a revised Newton's method. A forward–backward propagation scheme is first proposed for network computation of the Hessian matrix, $\mathbf{H}$, of the output error function of the MLP. A block Hessian matrix, $\mathbf{H}_b$, is then defined to approximate and simplify $\mathbf{H}$. Several lemmas and theorems are proved to uncover the important properties of $\mathbf{H}$ and $\mathbf{H}_b$, and verify the good approximation of $\mathbf{H}_b$ to $\mathbf{H}$; $\mathbf{H}_b$ preserves the major properties of $\mathbf{H}$. The theoretic analysis leads to the development of an efficient way for computing the inverse of $\mathbf{H}_b$ recursively. In the proposed second-order learning algorithm, the least squares estimation technique is adopted to further lessen the local minimum problems. The proposed algorithm overcomes not only the drawbacks of the standard backpropagation algorithm (i.e. slow asymptotic convergence rate, bad controllability of convergence accuracy, local minimum problems, and high sensitivity to learning constant), but also the shortcomings of normal Newton's method used on the MLP, such as the lack of network implementation of $\mathbf{H}$, ill representability of the diagonal terms of $\mathbf{H}$, the heavy computation load of the inverse of $\mathbf{H}$, and the requirement of a good initial estimate of the solution (weights). Several example problems are used to demonstrate the efficiency of the proposed learning algorithm. Extensive performance (convergence rate and accuracy) comparisons of the proposed algorithm with other learning schemes (including the standard backpropagation algorithm) are also made. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords: Multilayer perceptrons; Hessian matrix; Forward–backward propagation; Newton's method; Least squares estimation

1. Introduction

Gradient-descent-based backpropagation (BP) learning algorithm has been widely used for training multilayer perceptron (MLP) networks (Haykin, 1994; Lin and Lee, 1996). However, several drawbacks of the BP learning method have been observed; its convergence speed is usually too low, its convergence accuracy is hard to control, it is easily stuck in bad local minima, and the choice of proper learning constant largely depends on trial and error. Further numerical optimization theory (Luenberger, 1984) can be applied to overcome these drawbacks. One common approach is to upgrade the normal BP, which is a first-order learning algorithm, to a second-order one, the so-called Newton's method. Since Newton's method is an optimization algorithm with quadratic convergence speed (Luenberger, 1984), it can be used to improve the learning speed and accuracy of the normal BP. Also, since Newton's method is less sensitive to the learning constant, the choice of a proper learning constant is not difficult. However, several shortcomings of Newton's method, as mentioned later, make its use for training the MLP quite limited. This article aims at conquering these shortcomings, and develops an efficient second-order learning algorithm for the MLP.

There are several problems in using Newton's method to minimize the output error function of the MLP (Haykin, 1994; Lin and Lee, 1996).

1. Newton's method needs to compute the second-order derivatives of the output error function with respect to the network weights, i.e. the Hessian matrix. Since the computation of the Hessian matrix needs global information, Newton's method is not suitable for network computation. Besides, the large computation load of the Hessian matrix hinders its practical use in training MLPs.
2. One common strategy to simplifying the computation of Hessian matrix is to approximate the whole matrix by its diagonal terms only (we call it the diagonal Hessian matrix) (Battiti, 1992). This article will show that the

* Corresponding author. Tel: +886 3 5712121 ext 54315; Fax: +886 3 5715998; e-mail: ctlin@fnn.cn.nctu.edu.tw

diagonal Hessian matrix does not maintain the major properties of the true Hessian matrix of a MLP, and thus cannot be used to improve the convergence speed and accuracy of BP learning efficiently.

3. In using Newton's method for minimizing the output error of a MLP, each iteration requires the computation of the inverse of Hessian matrix, so the method is expensive in terms of both storage and computational requirements.
4. In order to converge, Newton's method requires a good initial estimate of the solution. This further restricts the practical usability of Newton's method on the MLP.

From these points, we know that standard Newton's method is not a practical technique for training the MLP. Although several alternatives or revised methods have been studied such as those based on conjugate-direction method and quasi-Newton method (Ricotti et al., 1988; Becker and LeCun, 1989; Makram-Ebeid et al., 1989; Bello, 1992), the aforementioned problems have not been seriously addressed and most problems still exist. Especially, in the existing second-order learning approaches, the computation-expensive nonlinear programming techniques in the numerical optimization theory are usually adopted, and the special properties of the Hessian matrix of the MLP are not taken into account to reduce the computation load.

In this article, we shall propose a novel second-order learning algorithm for the MLP, aimed at solving the four drawbacks of Newton's method. We first propose an order-based-derivative scheme (Werbos, 1990; Piche, 1994) to derive a network computation method for computing the Hessian matrix of a given MLP network. This can be viewed as a network implementation of the Hessian matrix. With this scheme, the computation of the Hessian matrix of a MLP can be performed as signals flow forward and backward in the MLP network. This solves the first drawback of Newton's method mentioned previously. From the proposed network computation method of the Hessian matrix, we can analyze the important properties of the Hessian matrix of the MLP easily; e.g. the Hessian matrix of a MLP is always a singular matrix. From such analysis, we clearly understand why the diagonal Hessian matrix is not a good approximation of the original Hessian matrix. This theoretically explains why the existing second-order learning algorithms based on the diagonal Hessian matrix do not display the expected advantages over Newton's method as mentioned in the second point of the last paragraph. To overcome this drawback, and avoid the large computation load of the whole Hessian matrix, we propose a better approximation of the Hessian matrix, called the block-diagonal Hessian matrix or block Hessian matrix, $\mathbf{H}_b$, for short. The block Hessian matrix keeps the singularity of the original Hessian matrix. Also, we show that the block Hessian matrix of a MLP is linearly dependent with the matrix formed by the gradients of the output error function E of this MLP, ∇E. This dependency property makes the equation, $\nabla E = 0$, the weight space have solutions even though $\mathbf{H}_b$ is singular, when we apply the block Hessian matrix ($\mathbf{H}_b$) for second-order approximation of the E function. Hence, like the original error function, there also exist extreme points for the error function which is second-order approximated by the block Hessian matrix.

Making use of the singularity and dependency property of the block Hessian matrix, we arrive at an efficient algorithm for solving the equation $\nabla E = 0$. This algorithm does not need the computation of the inverse of the block Hessian matrix explicitly, and thus solves the third drawback of Newton's method mentioned earlier. In the proposed algorithm, we also apply the least squares estimation technique (Goodwin and Sin, 1984; Stefanos and Anastassioy, 1988) to modify the original Newton's method. This further improves the convergence speed and accuracy of learning. Finally, since Newton's method only guarantees finding the extreme points of error functions (i.e. the points that result in $\nabla E = 0$), which may be minimum or maximum points, we develop three protection methods in this article to prevent the proposed second-order learning algorithm from converging in wrong directions. Among these three protection methods, two methods try to change the gradient of the error surface in the transient region, and keep the gradients in the steady states unchanged. These protection methods make Newton's method insensitive to initial states, and solve the fourth drawback of Newton's method.

The rest of this article is organized as follows. Section 2 derives a forward–backward propagation scheme for computing the Hessian matrix $\mathbf{H}$ of the MLP in the form of network operations. This section also defines the block Hessian matrix $\mathbf{H}_b$, and shows the singularity of $\mathbf{H}$ and $\mathbf{H}_b$, and the linear dependency of $\mathbf{H}_b$ and ∇E. In Section 3, the special properties of the block Hessian matrix are adopted to develop an efficient algorithm that greatly reduces the computation load of the inverse Hessian matrix. This section also uses the least squares estimation technique to increase learning speed and accuracy. Extensive computer simulations and performance comparisons with normal BP and diagonal Hessian matrix approaches are made in Section 4. In this section, three protection methods are given to further improve the proposed second-order learning algorithm. Conclusion and discussion are presented in Section 5.

2. Block Hessian matrix of multilayer perceptrons

2.1. *Network computation of Hessian matrix based on order-based derivative*

The gradient-descent-based BP learning method was widely used for training the MLP. Although the gradient-descent (or steepest-descent) method is one of the simplest optimization techniques, it is not a very effective one. Numerical optimization theory provides a rich and robust set of techniques which can be applied to neural networks to improve learning rates. The gradient-descent method considers only the first-order derivative of an error function. It is helpful to take into account higher-order derivatives. Using Taylor's series expansion on the error function $E(\mathbf{w})$ of a MLP around the current point $\mathbf{w}_0$ in the weight space, we have

$$E(\mathbf{w}) = E(\mathbf{w}_0) + (\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}\nabla E(\mathbf{w}_0) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}H(\mathbf{w}_0)(\mathbf{w}-\mathbf{w}_0) + \cdots, \qquad (1)$$

where $\mathbf{H}(\mathbf{w}_0)$ is called Hessian matrix and is the second-order derivative evaluated at $\mathbf{w}_0$:

$$\mathbf{H}(\mathbf{w}_0) \equiv \nabla^2 E(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_0} \text{ or } \mathbf{H}_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \qquad (2)$$

To find the minimum of $E(\mathbf{w})$, we set its gradient to zero:

$$\nabla E(\mathbf{w}) = \nabla E(\mathbf{w}_0) + H(\mathbf{w}_0)(\mathbf{w}-\mathbf{w}_0) + \cdots = 0 \qquad (3)$$

If we ignore the third- and higher-order terms, we obtain

$$\mathbf{w} = \mathbf{w}_0 - \mathbf{H}^{-1}(\mathbf{w}_0)\nabla E(\mathbf{w}_0) \qquad (4)$$

or using k to indicate the kth updating step, we obtain

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \mathbf{H}^{-1}(\mathbf{w}^{(k)})\nabla E(\mathbf{w}^{(k)}) \qquad (5)$$

This is called Newton's method of weight updating. Newton's method uses the second-order derivative in addition to the gradient to determine the next updating direction and step size. It can converge quadratically when close to a solution of a convex function. However, there are several drawbacks for Newton's method as mentioned in the last section. In this subsection, we shall derive a network computation scheme with forward–backward signal propagation to simplify the computation of Hessian matrix of the MLP. We shall also study the important properties of Hessian matrix of the MLP, and then propose an approximated matrix that keeps the important properties of the original one.

Consider a MLP network with L layers. For notation clarity, let $\#l$ denote the number of nodes in the lth layer of the MLP for $l = 1,2,\cdots,L$. The input of the jth node in the $(l+1)$th layer is represented by $\mathrm{net}_j^{(l+1)}$, and the output by

$$z_j^{(l+1)} \equiv f(\mathrm{net}_j^{(l+1)}) \equiv f\left(\sum_{k=1}^{\#l} w_{jk}^{(l)} z_k^{(l)}\right) \qquad (6)$$

where $f(\cdot)$ is an activation function, and $w_{jk}^{(l)}$ is the connection weight between the kth node in layer l and the jth node in layer $l+1$. The output error function of the MLP is defined by

$$E = \frac{1}{2}\sum_{j=1}^{\sharp L}(z_j^{(L)} - d_j)^2 \qquad (7)$$

where $z_j^{(L)}$ is the jth network output for the current input and d_j is the corresponding desired output.

We shall then use order-based derivative (Werbos, 1990; Piche, 1994) to derive the Hessian matrix of E with respect to connection weights $w_{ji}^{(l)}$ and $w_{mn}^{(p)}$ according to Eq. (2). We shall see that the use of order-based derivative can lead to a network implementation of the Hessian matrix; i.e. the computation of the Hessian matrix can be proceeded through the forward and backward data flow in a MLP network. Order-based derivative is defined as the deviation of a function, $E(z_1,z_2,\ldots,z_n)$, with respect to the deviations of a set of variables, $\{z_1,z_2,\ldots,z_n\}$, where this set of variables form a order set; i.e. any variable, z_j, is a function of variables $\{z_1,z_2,\ldots,z_{j-1}\}$. Due to the nest relationship in this set of variables, the total derivative of $E(z_1,z_2,\ldots,z_n)$ with respect to z_j, $j = 1,2,\ldots,n-1$ can be obtained recursively, based on the order relationship of the set of dependent variables, $\{z_1,z_2,\ldots,z_n\}$. The approach to such a recursive computation is called an order-based-derivative scheme. The reader is referred to (Piche, 1994) for a detailed example. According to the types of ordering, two kinds of order-based derivatives, forward and backward, can be distinguished. The concept of an order-based derivative is similar to that of a partial derivative formed by chain rule with ordinary derivatives. However, the former uses more precise notation to distinguish the value and ordering of a derivative, which are easily mixed up in the notation of partial derivative. Hence, as compared with using chain rule to derive first-order derivatives in a backpropagation algorithm, the concept of order-based derivative is more suitable for deriving complex and higher-order derivatives, such as the second-order derivatives in the Hessian matrix of a MLP.

At first, we use the backward scheme to derive the first-order order-based derivative of E with respect to w_{ji}:

$$\frac{\partial^+ E}{\partial w_{ji}^{(l)}} = \frac{\partial z_j^{(l+1)}}{\partial w_{ji}^{(l)}}\frac{\partial^+ E}{\partial z_j^{(l+1)}} = (f'(\mathrm{net}_j^{(l+1)})z_i^{(l)})\frac{\partial^+ E}{\partial z_j^{(l+1)}} \qquad (8)$$

where $l = 1,2,\cdots,L-1$, and $(\partial^+ E)/(\partial z_j^{(l+1)})$ is computed by the following backpropagation rule

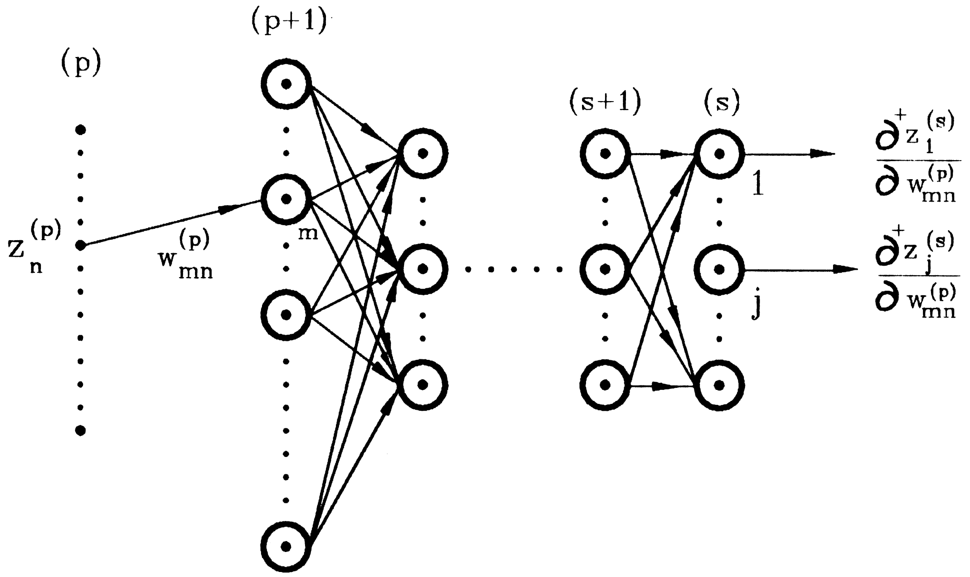
$$\frac{\partial^+ E}{\partial_j z^{(s)}} = \frac{\partial E}{\partial z_j^{(s)}} + \sum_{k=1}^{\#(s+1)}\frac{\partial z_k^{(s+1)}}{\partial z_j^{(s)}}\frac{\partial^+ E}{\partial z_k^{(s+1)}} \qquad (9)$$

where $s = L,\cdots,(l+1)$ and $j = 1,\cdots,\#s$. Notice that $(\partial z_k^{(s+1)})/(\partial z_j^{(s)}) = 0$ if $s = L$, and $(\partial \mathrm{E})/(\partial z_j^{(s)}) = 0$ if $s \neq L$.

Next, we find the second-order order-based derivative of E with respect to $w_{ji}^{(l)}$ and another connection weight $w_{mn}^{(p)}$ by computing the order-based derivative of Eq. (8) with respect to $w_{mn}^{(p)}$, assuming $p \le l$:

$$\frac{\partial^+}{\partial w_{mn}^{(p)}}\frac{\partial^+}{\partial w_{ji}^{(l)}}E = \frac{\partial^+}{\partial w_{mn}^{(p)}}(f'(\mathrm{net}_j^{(l+1)})z_i^{(l)})\frac{\partial^+ E}{\partial z_j^{(l+1)}} + \frac{\partial^+}{\partial w_{mn}^{(p)}}\left(\frac{\partial^+ E}{\partial z_j^{(l+1)}}\right)\cdot f'(\mathrm{net}_j^{(l+1)})\cdot z_i^{(l)} \qquad (10)$$

where the term $[(\partial^+)/(\partial w_{mn}^{(p)})][(\partial^+ E)/(\partial z_j^{(l+1)})]$ can be computed by the order-based derivative of Eq. (9) and has the



(p)
(p+1)
(s+1)
(s)
$z_n^{(p)}$
$w_{mn}^{(p)}$
m
1
j
$\frac{\partial^+ z_1^{(s)}}{\partial w_{mn}^{(p)}}$
$\frac{\partial^+ z_j^{(s)}}{\partial w_{mn}^{(p)}}$

Fig. 1. Network for computing the order-based derivatives $(\partial^+ z_j^{(s)})/(\partial w_{mn}^{(p)})$ using the forward propagation rule, where each node performs the operation, $\odot$, which computes the product net$\cdot f'$(net), where net is the net input of the node.

following backward form

$$\frac{\partial^+}{\partial w_{mn}^{(p)}}\left(\frac{\partial^+ E}{\partial z_j^{(s)}}\right) = \sum_{k=1}^{\#(s+1)} \frac{\partial z_k^{(s+1)}}{\partial z_j^{(s)}} \frac{\partial^+}{\partial w_{mn}^{(p)}}\left(\frac{\partial^+ E}{\partial z_k^{(s+1)}}\right) + \sum_{k=1}^{\#(s+1)} \frac{\partial^+}{\partial w_{mn}^{(p)}}(f'(\mathrm{net}_k^{(s+1)}))w_{kj}^{(s)} \frac{\partial^+ E}{\partial z_k^{(s+1)}} \tag{11}$$

where $s = L - 1,\cdots,(l + 1)$ and $j = 1,\cdots,\#s$. Notice that the initial state of Eq. (11) at $s = L$ is $[(\partial^+)/(\partial w_{mn}^{(p)})][(\partial^+ E)/(\partial z_j^{(L)})] = [(\partial^+)/(\partial w_{mn}^{(p)})](z_j^{(L)})$.

The first term of the right-hand side (RHS) of Eq. (11) is the same as the formula in the normal backpropagation algorithm, whereas the second term can be viewed as the bias term of the *j*th node in the *s*th layer of a MLP. If the differential of the activation function *f* with respect to its net input can be expressed as a function of its output $z_j^{(s)}$, then calculating the bias term in Eq. (11) needs only computation of the order-based derivative $[(\partial^+)/(\partial w^{(p)})]z_j^{(s)}$. For example, if $f(\mathrm{net}) = [2/(1+\exp(-\lambda\cdot\mathrm{net}))] - 1 = z_j$ (i.e. sigmoidal function), then $f'(\mathrm{net}) = (\lambda/2)(1 - z_j^2)$, and $[(\partial^+)/(\partial w)]f'(\mathrm{net}) = -\lambda z_j[(\partial^+ z_j)/(\partial w)]$. Hence to compute Eq. (11) in backward direction, we need to compute $[(\partial^+)/(\partial w_{mn}^{(p)})]z_j^{(s)}$, for $s = L, L-1,\cdots, p+1$. We shall then derive an algorithm for computing the bias term in Eq. (11) in the form of network operations like the backpropagation algorithm. Since this algorithm performs forward propagation on the network, it is called the forward propagation rule. This forward propagation rule can also be used to compute the first term of the RHS of Eq. (10).

At first, we have

$$\frac{\partial^+ z_j^{(s)}}{\partial w_{mn}^{(p)}} = \sum_{k=1}^{\#(s-1)} \frac{\partial z_j^{(s)}}{\partial z_k^{(s-1)}} \frac{\partial^+ z_k^{(s-1)}}{\partial w_{mn}^{(p)}} + \frac{\partial z_j^{(s)}}{\partial w_{mn}^{(p)}} = \sum_{k=1}^{\#(s-1)} w_{jk} f'(\mathrm{net}_j^{(s)}) \frac{\partial^+ z_k^{(s-1)}}{\partial w_{mn}^{(p)}} + \frac{\partial z_j^{(s)}}{\partial w_{mn}^{(p)}} \tag{12}$$

where $p + 1 < s \le L$. Notice that if $s = p + 1$ then Eq. (12) becomes

$$\frac{\partial^+ z_j^{(p+1)}}{\partial w_{mn}^{(p)}} = \frac{\partial z_j^{(p+1)}}{\partial w_{mn}^{(p)}} = \begin{cases} z_n^{(p)} & m = j \\ 0 & m \neq j, \end{cases} \tag{13}$$

and if $s \neq p + 1$ then the term, $(\partial z_j^{(s)})/(\partial w_{mn}^{(p)})$ in Eq. (12) is zero.

The formula in Eq. (12) can be represented in a network-operation form to simplify the computation. Consider a node in a MLP whose output is $z_j^{(s)}$. The order-based derivative of this node's output with respect to the connection weight $w_{nm}^{(p)}$ in layer *p*, i.e. $(\partial^+ z_j^{(s)})/(\partial w_{mn}^{(p)})$, is equal to the output activation value of the network shown in Fig. 1, when we let the inputs, $\mathrm{net}_m^{(p+1)} = 1$ and $\mathrm{net}_j^{(p+1)} = 0, j \neq m$, forward propagate through the network. The network shown in Fig. 1 has the same topology as the original MLP whose Hessian matrix is to be computed, whereas their node functions are different as indicated in Fig. 1. From Eq. (12) and Fig. 1, we find that each forward propagation starting from the *m*th node of the $(p+1)th$ layer can compute all the terms of $(\partial z_j^{(s)})/(\partial w_{mn}^{(p)})$, for $n = 1,\cdots,\#p$, where $p + 1 \le s \le L$. Moreover, for a specific $z_j^{(s)}$, the vector $[(\partial^+ z_j^{(s)})/(\partial w_{mn}^{(p)})]_{n=1,\cdots,\#p}$ is proportional to $[z_n^{(p)}]_{n=1,\cdots,\#p}$. The forward propagation rule in Eq. (12) can be used to compute the first term of

Table 1
The sizes of the networks to be propagated and the obtained columns of Hessian matrix for each forward–backward propagation on the MLP, $N_{5,3,3,1}$

Time	Subnetwork size for forward–backward propagation	Columns computed in Hessian matrix
1st time	$(1 \rightarrow 3 \rightarrow 1) + (3 \leftarrow 3 \leftarrow 1)$	a_1: 5 columns
2nd time	$(1 \rightarrow 3 \rightarrow 1) + (3 \leftarrow 3 \leftarrow 1)$	a_2: 5 columns
3rd time	$(1 \rightarrow 3 \rightarrow 1) + (3 \leftarrow 3 \leftarrow 1)$	a_3: 5 columns
4th time	$(1 \rightarrow 1) + (3 \leftarrow 1)$	b_1: 3 columns
5th time	$(1 \rightarrow 1) + (3 \leftarrow 1)$	b_2: 3 columns
6th time	$(1 \rightarrow 1) + (3 \leftarrow 1)$	b_3: 3 columns
7th time	$(1) + (1)$	c_1: 3 columns

Here, considering the first forward–backward propagation for example, $(1 \rightarrow 3 \rightarrow 1) + (3 \leftarrow 3 \leftarrow 1)$ means the signals flow forward from a layer-two node to the three layer-three nodes, and to the single output node in layer four, and then the signals flow backward from the single output node to the three layer-three nodes, and finally to the three layer-two nodes. The second and third cycles of forward–backward propagation are performed on similar subnetworks, but they start at a different node in layer two.

the RHS of Eq. (10) and the second term of the RHS of Eq. (11).

The computations of Eqs. (10)–(12) are summarized as follows. In computing the element of Hessian matrix $(\partial^+\partial^+E)/(\partial w_{mn}^{(p)}\partial w_{ji}^{(l)})$, we perform one forward propagation on the network in Fig. 1 starting from the mth node in layer $p+1$ to obtain all the bias terms in Eq. (11) and the first term of the RHS of Eq. (10). Then according to Eq. (11) and then Eq. (10), we perform one backward propagation process on the original network to find all the terms in the column of the Hessian matrix right below the term of $(\partial^+\partial^+E)/(\partial w_{mn}^{(p)}\partial w_{ji}^{(p)})$; i.e. the terms $(\partial^+\partial^+E)/(\partial w_{mn}^{(p)}\partial w_{ji}^{(l)})$ for all $l \geq p$. Repeat this process until all such terms in all the columns of the Hessian matrix are obtained. Finally, we can apply the symmetric property and the column proportional property of the Hessian matrix of the MLP shown in the following lemma to get all the other terms and obtain the complete Hessian matrix. For a MLP with N nodes, its whole Hessian matrix can be obtained after only $(N-k_1)$ cycles of forward and backward propagation, where k_1 is the number of input nodes. The forward–backward propagation corresponding to the pth layer will produce $\#(p+1)\cdot\#p$ columns in the Hessian matrix, in which $\#(p+1)$ columns are linearly independent (as being shown in the next subsection). Moreover, the network for a forward–backward propagation will be getting smaller; i.e. the computation load is lessening as the forward–backward propagation process proceeds.

As an example, in computing the Hessian matrix of a four-layer fully-connected MLP with structure $N_{5,3,3,1}$ (meaning that the node number of each layer is 5, 3, 3, 1, respectively, from the input layer), the sizes of the networks to be propagated and the obtained columns of Hessian matrix for each cycle of forward–backward propagation are listed in Table 1. Correspondingly, the form of the

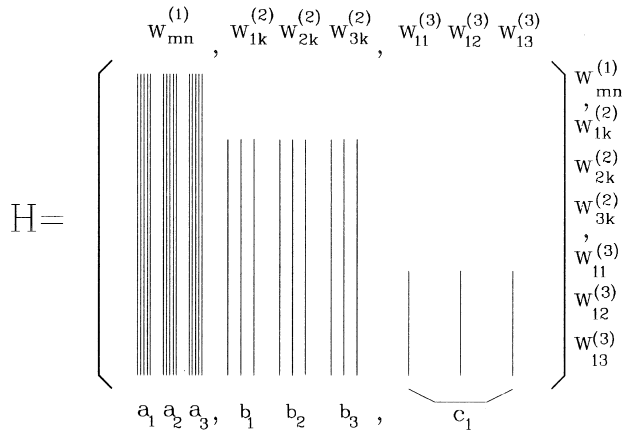

$W_{mn}^{(1)}$, $W_{1k}^{(2)}$ $W_{2k}^{(2)}$ $W_{3k}^{(2)}$, $W_{11}^{(3)}$ $W_{12}^{(3)}$ $W_{13}^{(3)}$
H=
$W_{mn}^{(1)}$, $W_{1k}^{(2)}$ $W_{2k}^{(2)}$ $W_{3k}^{(2)}$, $W_{11}^{(3)}$ $W_{12}^{(3)}$ $W_{13}^{(3)}$
a_1 a_2 a_3 , b_1 b_2 b_3 , c_1

Fig. 2. Form of resultant Hessian matrix after seven times of forward–backward propagation on the MLP, $N_{5,3,3,1}$, where $m = 1,2,3$, $n = 1,2,\ldots,5$, and $k = 1,2,3$.

resultant Hessian matrix after the seven cycles of forward–backward propagation is shown in Fig. 2. In Fig. 2, columns a_1, a_2, a_3, b_1, b_2, b_3, and c_1 correspond to the first, second, $\cdots$, and seventh cycle of forward–backward propagation, respectively. This correspondence is also shown in Table 1. More clearly, the first cycle of forward–backward propagation produces the five columns of Hessian matrix indicated by a_1 in Fig. 2. These five columns consist of the order-based derivatives, $(\partial^+\partial^+E)/(\partial w_{1n}^{(1)}\partial w_{ji}^{(p)})$, for $n = 1,2,\ldots,5$, where $\{w_{ji}^{(p)}\} = \{w_{11}^{(1)},\ldots,w_{35}^{(1)},w_{11}^{(2)},\ldots,w_{33}^{(2)},w_{11}^{(3)},w_{12}^{(3)},w_{13}^{(3)}\}$ i.e. those weights shown on the right-hand side of the Hessian matrix in Fig. 2. Similarly, the second and third cycles of forward–backward propagation produce those columns indicated by a_2 and a_3 in Fig. 2, respectively, each set including five columns. They consist of the order-based derivatives, $(\partial^+\partial^+E)/(\partial w_{2n}^{(1)}\partial w_{ji}^{(p)})$ and $\partial^+\partial^+E/\partial w_{3n}^{(1)}\partial w_{ji}^{(p)}$, for $n = 1,2,\ldots,5$. The fourth cycle of forward–backward propagation produces the three columns of Hessian matrix indicated by b_1 in Fig. 2. These three columns consist of the order-based derivatives, $(\partial^+\partial^+E)/(\partial w_{1k}^{(2)}\partial w_{ji}^{(p)})$, for $k = 1,2,3$, where $\{w_{ji}^{(p)}\} = \{w_{11}^{(1)},\ldots,w_{33}^{(2)},w_{11}^{(3)},w_{12}^{(3)},w_{13}^{(3)}\}$,i.e. those weights shown on the corresponding right-hand side of the Hessian matrix in Fig. 2. Similarly, the fifth and sixth cycles of forward–backward propagation produce those columns indicated by b_2 and b_3 in Fig. 2, respectively, each set including three columns. They consist of the order-based derivatives, $(\partial^+\partial^+E)/(\partial w_{2k}^{(2)}\partial w_{ji}^{(p)})$ and $(\partial^+\partial^+E)/(\partial w_{3k}^{(2)}\partial w_{ji}^{(p)})$, for $k = 1,2,3$. Finally, the seventh cycle of forward–backward propagation produces the three columns of Hessian matrix indicated by c_1 in Fig. 2. These three columns consist of the order-based derivatives, $(\partial^+\partial^+E)/(\partial w_{1k}^{(3)}\partial w_{ji}^{(p)})$, for $k = 1,2,3$, where $\{w_{ji}^{(p)}\} = \{w_{11}^{(3)},w_{12}^{(3)},w_{13}^{(3)}\}$. Hence, in Fig. 2, all the elements in the columns indicated by a_1, a_2, a_3, b_1, b_2, b_3, and c_1 are computed by seven cycles of forward–backward propagation. The rest of the elements of the Hessian matrix (i.e. the upper-left empty part in the Hessian matrix shown in Fig. 2) can be obtained directly from the computed elements by using the symmetric property of Hessian matrix shown in Section 2.2.

2.2. Block Hessian matrix and its properties

In this subsection, we shall study some properties of the Hessian matrix of the MLP based on the order-based-derivative formulas derived in the last subsection. Consider a MLP network with L layers, each layer containing an extra node (called 'threshold node') with fixed activation value -1 to provide the threshold value for each node in the next layer. At first, we show that the Hessian matrix derived from the order-based-derivative scheme in the last subsection keeps the symmetric property of a Hessian matrix.

Lemma 1. The matrix whose elements are given in Eq. (10) is symmetric, i.e.

$$\frac{\partial^+}{\partial w_{mn}^{(p)}}\frac{\partial^+}{\partial w_{ji}^{(l)}}E(\mathbf{z}^{(L)})=\frac{\partial^+}{\partial w_{ji}^{(l)}}\frac{\partial^+}{\partial w_{mn}^{(p)}}E(\mathbf{z}^{(L)}) \tag{14}$$

where $\mathbf{z}^{(L)}$ is the output vector of the MLP.

Proof. Let $\tilde{E}(\mathbf{w}^{(1)},\cdots,\mathbf{w}^{(L)})$ represent the error function $E(\mathbf{z}^{(L)})$ expressed in terms of connection weight vectors, $\mathbf{w}^{(s)}\equiv[w_{ji}^{(s)}]$, for $s=1,\cdots,L$. According to the concept of order-based derivative, we have

$$\frac{\partial^+E(\mathbf{z}^{(L)})}{\partial w_{ji}^{(l)}}=\frac{\partial}{\partial w_{ji}^{(l)}}\tilde{E}(\mathbf{w}^{(1)},\cdots,\mathbf{w}^{(L)}) \tag{15}$$

and

$$\frac{\partial^+}{\partial w_{mn}^{(p)}}\frac{\partial^+}{\partial w_{ji}^{(l)}}E(\mathbf{z}^{(L)})=\frac{\partial^2}{\partial w_{mn}^{(p)}\partial w_{ji}^{(l)}}\tilde{E}(\mathbf{w}^{(1)},\cdots,\mathbf{w}^{(L)}) \tag{16}$$

Similarly, we can obtain

$$\frac{\partial^+}{\partial w_{ji}^{(l)}}\frac{\partial^+}{\partial w_{mn}^{(p)}}E(\mathbf{z}^{(L)})=\frac{\partial^2}{\partial w_{ji}^{(l)}\partial w_{mn}^{(p)}}\tilde{E}(\mathbf{w}^{(1)},\cdots,\mathbf{w}^{(L)}) \tag{17}$$

Since the derivative ordering is changeable for partial derivative,Eq. (16) is equal to Eq. (17), i.e.

$$\frac{\partial^+}{\partial w_{mn}^{(p)}}\frac{\partial^+}{\partial w_{ji}^{(l)}}E(\mathbf{z}^{(L)})=\frac{\partial^+}{\partial w_{ji}^{(l)}}\frac{\partial^+}{\partial w_{mn}^{(p)}}E(\mathbf{z}^{(L)}) \tag{18}$$

This completes the proof.

Notice that the change of derivative ordering in Lemma 1 (see Eq. (14) might not be allowed by other forms of derivatives, e.g. $[(\partial^+/\partial z_i)(\partial/z_j)]E\neq[(\partial/\partial z_j)(\partial^+/\partial z_i)]E$ and $[\partial^+/(\partial w_{mn}^{(p)})][\partial^+/(\partial z_j^{(p)})]E\neq[\partial^+/(\partial z_j^{(p)})][\partial^+/(\partial w_{mn}^{(p)})]E$.

The following theorem shows an important property of the Hessian matrix of the MLP.

Theorem 1. Consider a MLP network whose error function is $E=(1/2)\sum_{i=1}^{\#L}(z_i^{(L)}-d_i)^2$. Assume the differential of the node activation function of the MLP can be expressed as a function of the node output (e.g. assume the activation function is a sigmoidal function). Then the Hessian matrix of E with respect to the connections weights of the MLP is a singular matrix.

Proof. We shall first show that for a fixed m, we have $[\partial^+/(\partial w_{mn}^{(1)})][(\partial^+E)/(\partial w_{ji}^{(l)})]=\Delta_{ji}^l z_n^{(1)}$, where Δ_{ji}^l is a constant determined by the forward–backward propagation computation between layer 2 and layer $l+1$. From the forward propagation rule in Eq. (12), we know that $(\partial^+z_j^{(s)})/(\partial w_{mn}^{(1)})$ is proportional to $z_n^{(1)}$, and the proportional constant, say $c_j^{(s)}$, is dependent on the node output in concern, i.e. $z_j^{(s)}$. Hence we have $(\partial^+z_j^{(s)})/(\partial w_{mn}^{(1)})=c_j^{(s)}z_n^{(1)}$. Considering Eq. (11) for computing $[\partial^+/(\partial w_{mn}^{(1)})][(\partial^+E)/(\partial z_j^{(s)})]$, since $f'(\text{net}_k^{(s+1)})$ is a function of $z_k^{(s+1)}$, the second term of the RHS of Eq. (11) is proportional to $z_n^{(1)}$. Also, since $\partial^+/\partial w_{mn}^{(1)}(\partial^+E/\partial z_j^{(L)})$ is proportional to $z_n^{(1)}$, the first term of the RHS of Eq. (11) is also proportional to $z_n^{(1)}$. Hence the term $\partial^+/\partial w_{mn}^{(1)}(\partial^+E/\partial z_j^{(s)})$ computed in Eq. (11) is proportional to $z_n^{(1)}$. This results in the fact that the second term of the RHS of Eq. (10) is proportional to $z_n^{(1)}$, when Eq. (10) is used to compute the term $[\partial^+/(\partial w_{mn}^{(1)})][(\partial^+E)/(\partial w_{ji}^{(l)})]$. Since the first term of the RHS of Eq. (10) is also proportional to $z_n^{(1)}$, we conclude that

$$\frac{\partial^+}{\partial w_{mn}^{(1)}}\frac{\partial^+E}{\partial w_{ji}^{(l)}}=\Delta_{ji}^l z_n^{(1)}, \tag{19}$$

where Δ_{ji}^l is a constant determined by the forward–backward propagation computation between layer 2 and the jth node of layer $l+1$.

With the same reason as in the above analysis, we can prove that the next column of the Hessian matrix associated with the same m has the same property:

$$\frac{\partial^+}{\partial w_{mn+1}^{(1)}}\frac{\partial^+E}{\partial w_{ji}^{(l)}}=\Delta_{ji}^l z_{n+1}^{(1)} \tag{20}$$

for all $l=1,\cdots,L-1$ and $i,j=1,\cdots,\#l$. From Eqs. (19) and (20), we find that there exist at least two columns in the Hessian matrix which are linearly dependent, so the Hessian matrix is singular. This completes the proof.

In the above theorem, we assume there are at least two input nodes for a MLP. This assumption is always true if a threshold node is added to each layer of the MLP. Many existing literatures point out that the computation of the Hessian matrix of a MLP is quite complex (Becker and LeCun, 1989; LeCun, 1989), partly because of the large size of the Hessian matrix, and to the lack of its network implementation (such as the form we derived earlier). To simplify the computation, the diagonal terms of the Hessian matrix were usually used to approximate the whole Hessian matrix. From the aforementioned analysis, we clearly see that such approximation is improper, since the singular property of the true Hessian matrix of a MLP is not preserved by the approximating diagonal Hessian matrix. This also explains why Newton's method based on the diagonal Hessian matrix cannot speed up the weight convergence in MLP learning efficiently (Battiti, 1992). The network implementation of the Hessian matrix proposed in the last subsection dose make its computation easier. To further simplify the computation, we shall next derive a block-diagonal Hessian matrix or block Hessian matrix, $\mathbf{H}_{\mathrm{b}}$, for short, to approximate the true Hessian matrix, $\mathbf{H}$. In the block Hessian matrix, we consider only the second-order order-based derivatives of the error function with respect to the two weights in the same layer, $(\partial^+\partial^+E)/(\partial w_{mn}^{(p)}\partial w_{ji}^{(p)})$, and let the other second-order order-based derivatives to zero, i.e. $(\partial^+\partial^+E)/(\partial w_{mn}^{(p)}\partial w_{ji}^{(l)}) = 0$ for $p \neq l$. This reduces the computation load of the Hessian matrix greatly. More importantly, the block Hessian matrix keeps the singularity property of the true Hessian matrix as shown in the following.

Consider a MLP with a threshold node in each layer. The elements of the block Hessian matrix of the MLP are arranged in the order of layer numbers and indexes of weights such that the derivatives with respect to the weights in the same layer flock together in a block lying along the diagonal of the Hessian matrix, i.e.

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{\mathrm{b}}^{(1)} & * & * & * & * \\ * & \ddots & * & * & * \\ * & * & \mathbf{H}_{\mathrm{b}}^{(p)} & * & * \\ * & * & * & \ddots & * \\ * & * & * & * & \mathbf{H}_{\mathrm{b}}^{(L-1)} \end{bmatrix},$$

$$\mathbf{H}_{\mathrm{b}} \equiv \begin{bmatrix} \mathbf{H}_{\mathrm{b}}^{(1)} & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & \mathbf{H}_{\mathrm{b}}^{(p)} & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \mathbf{H}_{\mathrm{b}}^{(L-1)} \end{bmatrix} \quad (21)$$

where $\mathbf{H}_{\mathrm{b}}^{(p)}$ with dimension $(\#(p+1)-1)(\#p) \times (\#(p+1)-1)(\#p)$ is the block corresponding to the pth layer of the MLP. Notice that '-1' in the term $(\#(p+1)-1)$ is due to the threshold node we assumed in each layer except the output layer. Hence for the output layer, '-1' should be taken out; i.e. the dimension of $\mathbf{H}_{\mathrm{b}}^{(L-1)}$ is $(\#L)(\#(L-1)) \times (\#L)(\#(L-1))$. This notation simplification will also be used throughout the following development.

The following theorem shows that the block Hessian matrix of a MLP is singular.

Theorem 2. Under the same assumptions in Theorem 1, each submatrix $\mathbf{H}_{\mathrm{b}}^{(p)}$ of the block Hessian matrix $\mathbf{H}_{\mathrm{b}}$ is a singular matrix with $rank(\mathbf{H}_{\mathrm{b}}^{(p)}) \leq \#(p+1)-1$.

Proof. As we noted in the above that $\mathbf{H}_{\mathrm{b}}^{(p)}$ is a $(\#(p+1)-1)(\#p) \times (\#(p+1)-1)(\#p)$ matrix. Each element of $\mathbf{H}_{\mathrm{b}}^{(p)}$ is computed by Eq. (10). To obtain a set of columns of $\mathbf{H}_{\mathrm{b}}^{(p)}$ corresponding to some fixed m, for $n = 1,2,\ldots,\#p$, we need to compute the term $\partial^+/\partial w_{mn}^{(p)}(f'(net_k^{(s)}))$, for $s = p+1,\ldots,L$, which requires the computation of the order-based derivative $(\partial^+ z_j^{(s)})/(\partial w_{mn}^{(p)})$ assuming the sigmoidal activation function is used [see the first term of the RHS of Eq. (10) and the second term of the RHS of Eq. (11)]. From the forward propagation rule in Eq. (12) with initial value $z_n^{(p)}$, we have

$$\frac{\partial^+ z_j^{(s)}}{\partial w_{mn}^{(p)}} = c_j^{(s)} z_n^{(p)} \text{ for all } p+1 \leq s \leq L \quad (22)$$

where $c_j^{(s)}$ is a constant determined by the connection weights between the node with output $z_j^{(p+1)}$ and the node with output $z_m^{(s)}$, and is thus not a function of $z_n^{(p)}$. Hence, when we compute some specific row of $\mathbf{H}_{\mathrm{b}}^{(p)}$ [i.e. the indexes i and j are fixed in Eqs. (10)–(12)], the first term of the RHS of Eq. (10) is proportional to $z_n^{(p)}$.

We shall next show that the second term of Eq. (10) is also proportional to $z_n^{(p)}$ when we compute some specific row of $\mathbf{H}_{\mathrm{b}}^{(p)}$. This is achieved by showing that the first and second terms of the RHS of Eq. (11) are both proportional to $z_n^{(p)}$. With the same reason in deriving Eq. (22), we see that the bias term [i.e. the second term of the RHS of Eq. (11)] is also proportional to $z_n^{(p)}$, and the proportion constant is determined by the forward–backward propagation computation between node m in layer $p+1$ to the specific row position. As to the first term of Eq. (11), it is obtained by the backpropagation rule with initial value

$$\frac{\partial^+}{\partial w_{mn}^{(p)}}\left(\frac{\partial^+ E}{\partial z_j^{(L)}}\right) = \sum_{k=1}^{\#L} \frac{\partial^+}{\partial w_{mn}^{(p)}}(d_k - z_k^{(L)}) \quad (23)$$

where d_k is the kth component of the desired output. According to Eq. (22), Eq. (23) is also proportional to $z_n^{(p)}$. Hence the first term of the RHS of Eq. (11) is also proportional to $z_n^{(p)}$.

This analysis proves that the order-based derivative computed by Eq. (11) and thus that computed by Eq. (10) are proportional to $z_n^{(p)}$. Hence for a specific m, the adjacent two columns of the $\mathbf{H}_{\mathrm{b}}^{(p)}$ matrix are proportional to $z_n^{(p)}$ and $z_{n+1}^{(p)}$, respectively. Since the full columns of $\mathbf{H}_{\mathrm{b}}^{(p)}$ are spanned by the vectors $[\partial^+/(\partial w_{mn}^{(p)})][\partial^+/(\partial \mathbf{w}^{(p)})E]$ and the earlier analysis shows that they are also spanned by the vectors $[\partial^+/(\partial w_{m1}^{(p)})][\partial^+/(\partial \mathbf{w}^{(p)})E]$, for $m = 1,\cdots,\#(p+1)-1$, we can conclude that $\mathrm{rank}(\mathbf{H}_{\mathrm{b}}^{(p)}) \leq \#(p+1)-1$. This completes the proof.

This theorem shows that each submatrix (block) of the block Hessian matrix is singular. This obviously implies that the whole block Hessian matrix is singular. Since the rank of $\mathbf{H}_{\mathrm{b}}^{(p)}$ is equal to the number of nodes in the layer next to the layer corresponding to $\mathbf{H}_{\mathrm{b}}^{(p)}$, the equality in Theorem 2 usually holds, i.e. $\mathrm{rank}(\mathbf{H}_{\mathrm{b}}^{(p)}) = \#(p+1)-1$. Further, according to the property in Theorem 2, we can decompose the $\mathbf{H}_\mathrm{b}^{(p)}$ matrix into $\#(p+1)-1$ submatrices, each with dimension $(\#(p+1)-1)(\#p) \times (\#p)$. Matrix $\mathbf{H}_\mathrm{b}^{(p)}$ can thus be represented by

$$\mathbf{H}_\mathrm{b}^{(p)} = \sum_{k=1}^{\#(p+1)-1} \boldsymbol{\eta}_k^{(p)} \left[\mathbf{0}_{k-1}, \mathbf{z}^{(p)}, \mathbf{0}_{\#(p+1)-1-k}\right] \tag{24}$$

where $\boldsymbol{\eta}_k^{(p)}$ is the vector derived by Eq. (11) and Eq. (12), $\mathbf{0}_j$ is a zero(row) vector containing $j\cdot(\#p)$ zeros, and $\mathbf{z}^{(p)}$ is the output (row) vector of layer p.

From numerical optimization theory (Luenberger, 1984), a system usually has no extreme points if its Hessian matrix is singular. However, since there are usually many extreme points in the error surface of a MLP, especially for large networks, we expect that $\mathbf{H}_\mathrm{b}$ and ∇E are linearly dependent. Otherwise, the $\mathbf{H}_\mathrm{b}$ matrix cannot be used for second-order approximation of the E function in finding the extreme points of E by Newton's method. The following theorem proves this property.

Theorem 3. Under the same assumptions in Theorem 1 and assuming that the rank of $\mathbf{H}_\mathrm{b}^{(p)}$ is $\#(p+1)-1$ and there is no zero element in $\nabla E^{(p)}$, then the rank of matrix $[\mathbf{H}_\mathrm{b}^{(p)}, \nabla E^{(p)}]$ is also equal to $\#(p+1)-1$, where $\nabla E^{(p)}$ is the gradient of E corresponding to layer p.

Proof. By the backpropagation rule, we can derive

$$\frac{\partial^+ E}{\partial w_{mn}^{(p)}} = c_m z_n^{(p)}, \tag{25}$$

where nonzero constant c_m is not a function of connection weights of layer p. For convenience, we let $s = \#(p+1)-1$.Then Eq. (24) can be rewritten as

$$\mathbf{H}_\mathrm{b}^{(p)} = \sum_{k=1}^{s} \boldsymbol{\xi}_k^{(p)} \left[\mathbf{0}_{k-1}, \left(\frac{\partial^+ E}{\partial \mathbf{w}_k^{(p)}}\right)^\mathrm{T}, \mathbf{0}_{s-k}\right] \tag{26}$$

where $\boldsymbol{\xi}_k^{(p)} = \boldsymbol{\eta}_k^{(p)}/c_k$, and $[(\partial^+ E)/(\partial w_k^{(p)})]^\mathrm{T} = [(\partial^+ E)/(\partial w_{k1}^{(p)}), \cdots, (\partial^+ E)/(\partial w_{kp}^{(p)})]$.

By the symmetric property of a Hessian matrix, we have

$$(\mathbf{H}_\mathrm{b}^{(p)})^\mathrm{T} = \mathbf{H}_\mathrm{b}^{(p)} = \sum_{k=1}^{s} \begin{bmatrix} \mathbf{0}_{k-1}^\mathrm{T} \\ \dfrac{\partial^+ E}{\partial w_k^{(p)}} \\ \mathbf{0}_{s-k}^\mathrm{T} \end{bmatrix} (\boldsymbol{\xi}_k^{(p)})^\mathrm{T} \tag{27}$$

Notice that the matrix

$$\mathbf{M} \equiv \begin{bmatrix} (\boldsymbol{\xi}_1^p)^\mathrm{T} \\ \vdots \\ (\boldsymbol{\xi}_s^p)^\mathrm{T} \end{bmatrix}$$

is a $s \times (\#p)(\#(p+1)-1)$ matrix. Since rank$(M) = \#(p+1)-1 = s$, there exists a series of column operations such that the $s \times 1$ unit vector, $[1_1, 1_2, \ldots, 1_s]^\mathrm{T}$, can be spanned by the columns of $\mathbf{M}$. Hence, the vector $\nabla E^{(p)} = \{[(\partial^+ E)/(\partial w_1^{(p)})], \cdots, [(\partial^+ E)/(\partial w_s^{(p)})]\}^\mathrm{T}$ can also be spanned by the columns of $\mathbf{H}_\mathrm{b}^{(p)}$, and thus rank$[\mathbf{H}_\mathrm{b}^{(p)}, \nabla E^{(p)}] = s$. This completes the proof.

This theorem shows that $\mathbf{H}_\mathrm{b}^{(p)}$ and $\nabla E^{(p)}$ are linearly dependent. This implies that the block Hessian matrix $\mathbf{H}_\mathrm{b}$ is linearly dependent with

$$\nabla E \equiv \begin{bmatrix} \nabla E^{(1)} \\ \vdots \\ \nabla E^{(L-1)} \end{bmatrix}$$

since $\mathbf{H}_\mathrm{b}$ is composed of $\mathbf{H}_\mathrm{b}^{(p)}$ as defined in Eq. (21). Theorems 2 and 3 show that the block Hessian matrix $\mathbf{H}_\mathrm{b}$ preserves the singularity and extreme-point properties of the true Hessian matrix $\mathbf{H}$. Hence we can approximate the error function of a MLP, $E \equiv -(1/2)\sum_{k=1}^{\#L}(z_k^{(L)} - d_k)^2$, by $\tilde{E}(\mathbf{w}) = E(\mathbf{w}_0) + \triangle\mathbf{w}^\mathrm{T}\nabla E(\mathbf{w}_0) + (1/2)\triangle\mathbf{w}^\mathrm{T}\mathbf{H}_\mathrm{b}\triangle\mathbf{w}$. In the next section, we shall derive an efficient second-order learning algorithm for the MLP by minimizing such an approximated error function.

3. Second-order learning algorithm based on block Hessian matrix

In this section, we shall develop a second-order learning algorithm for the MLP based on the block Hessian matrix. By making use of the properties of the block Hessian matrix, this algorithm can reduce the computation load of the block Hessian matrix, its inverse, and the process for finding the least squares solution.

3.1. Inverse of block Hessian matrix

In using Newton's method to minimize an error function E approximated by the block Hessian matrix, we need to solve the linear equations, $\nabla E = -\mathbf{H}_\mathrm{b}\triangle\mathbf{w}$. Since ∇E and $\mathbf{H}_\mathrm{b}$ are linear dependent as shown in the last section, we cannot compute the inverse of the block Hessian matrix, $\mathbf{H}_\mathrm{b}^{-1}$, directly to solve the linear equations, $\nabla E = -\mathbf{H}_\mathrm{b}\triangle\mathbf{w}$. To address this problem, we apply the Levenberg–Marquardt method to replace $\mathbf{H}_\mathrm{b}^{-1}$ by $(\lambda I + \mathbf{H}_\mathrm{b})^{-1}$ (LeCun, 1989; Silva and Almeida, 1990), where λ is an arbitrary small positive number. Hence, the problem now is to solve the linear equations, $\triangle\mathbf{w} = -(\lambda I + \mathbf{H}_\mathrm{b})^{-1}\nabla E$, i.e. $\triangle\mathbf{w}^{(p)} = -(\lambda I + \mathbf{H}_\mathrm{b}^{(p)})^{-1}\nabla E^{(p)}$, $p = 1,2,\ldots,L-1$. We shall next propose an algorithm to simplify the computation of $(\lambda I + \mathbf{H}_\mathrm{b}^{(p)})^{-1}$ based on the special property of the block Hessian matrix.

According to Eq. (24), the inverse matrix $(\lambda I + \mathbf{H}_b^{(p)})^{-1}$ can be represented by

$$(\lambda I + \mathbf{H}_b^{(p)})^{-1} = \left(\lambda I + \sum_{k=1}^{\#(p+1)-1} \boldsymbol{\eta}_k^{(p)} \left[\mathbf{0}_{k-1}, \mathbf{z}^{(p)}, \mathbf{0}_{\#(p+1)-1-k}\right]\right)^{-1} \tag{28}$$

for $p = 1,2,\ldots,L-1$. The inverse in Eq. (28) can be easily obtained by the matrix inverse lemma (Kailath, 1980). The matrix inverse lemma says if $\mathbf{A}$ is an invertible matrix, and $\mathbf{u}$ and $\mathbf{v}$ are two column vectors then the following equality holds,

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{(\mathbf{A}^{-1}\mathbf{u})(\mathbf{v}^T\mathbf{A}^{-1})}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}} \tag{29}$$

Using Eq. (29) recursively, we can find the $(\#(p+1)-1)(\#p) \times (\#(p+1)-1)(\#p)$ inverse matrix of Eq. (28) after $\#(p+1)-1$ iterations. It takes $(\#(p+1)-1)(\#p)$ iterations to compute such an inverse matrix in the normal way. Hence, by making use of the singular property of $\mathbf{H}_b^{(p)}$, we reduce the computation load of $(\lambda I + \mathbf{H}_b^{(p)})^{-1}$ by $\#p$ times.

Another advantage of this method is that the computation of the matrix inverse can be performed separately for different layers p, $p = 1, 2, \ldots, L-1$, due to its recursive form. This can shorten the duty cycle of computation. Hence, with the proposed computation method based on the matrix inverse lemma, the use of the block Hessian matrix for substituting the diagonal Hessian matrix in Newton's method can preserve the property of the true Hessian matrix at the expense of only small extra computation load.

3.2. Least squares update method

Consider that there are $\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(s)$ training patterns for training a MLP. The corresponding desired output vectors are $\mathbf{d}(1), \mathbf{d}(2), \cdots, \mathbf{d}(s)$. The total error function to be minimized is

$$E_{\text{total}} \equiv \frac{1}{2}\sum_{i=1}^{s} \|\mathbf{d}(i) - \mathbf{z}^{(L)}(i)\|^2 \tag{30}$$

where $\mathbf{z}^{(L)}(1), \mathbf{z}^{(L)}(2), \cdots, \mathbf{z}^{(L)}(s)$ are the network output vectors corresponding to the inputs $\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(s)$, respectively. The concept of standard Newton's method is to approximate E_{total} by a second-order function, $\tilde{E}_{\text{total}}$, as follows

$$\tilde{E}_{\text{total}}(\mathbf{w}(k) + \Delta\mathbf{w}) = E_{\text{total}}(\mathbf{w}(k)) + (\nabla E_{\text{total}})^T \Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^T \mathbf{H}_{\text{total}} \Delta w, \tag{31}$$

where $\mathbf{H}_{\text{total}}$ is the Hessian matrix of E_{total} taking value at weights $\mathbf{w}(k)$. Then the minimum points of E_{total} are obtained approximately by finding the minimum points of $\tilde{E}_{\text{total}}$ according to

$$\Delta\mathbf{w} = -\mathbf{H}_{\text{total}}^{-1} \nabla E_{\text{total}} \tag{32}$$

In using the above update rule, we usually find the average gradient value over s training patterns to get ∇E_{total}, and the average of s Hessian matrices corresponding to the s training patterns to get $\mathbf{H}_{\text{total}}$. From qualitative analysis, since this approach smooths the update direction and size corresponding to each training pattern, it cannot speed up the convergence speed efficiently.

We shall now adopt the least squares (LS) estimation technique to derive a fast update rule for $\Delta\mathbf{w}$. At first, we apply Newton's method on each individual training pattern and produce s gradient equations:

$$H(\mathbf{x}(i)|\mathbf{w}(k))\Delta\mathbf{w} = -\nabla E(\mathbf{x}(i)|\mathbf{w}(k)) \tag{33}$$

for $i = 1, 2, \ldots, s$. Then we solve the following combination of equations in the LS sense,

$$\begin{bmatrix} H(\mathbf{x}(1)|\mathbf{w}(k)) \\ \vdots \\ \vdots \\ H(\mathbf{x}(s)|\mathbf{w}(k)) \end{bmatrix} \Delta\mathbf{w} = -\begin{bmatrix} \nabla E(\mathbf{x}(1)|\mathbf{w}(k)) \\ \vdots \\ \vdots \\ \nabla E(\mathbf{x}(s)|\mathbf{w}(k)) \end{bmatrix} \tag{34}$$

to obtain

$$\Delta\mathbf{w} = -\left[H_1^T H_1 + \cdots + H_s^T H_s\right]^{-1}\left[H_1^T \nabla E_1 + \cdots + H_s^T \nabla E_s\right] \tag{35}$$

where $H_i = H(\mathbf{x}(i)|\mathbf{w}(k))$ and $\nabla E_i = E(\mathbf{x}(i)|\mathbf{w}(k))$. To solve Eq. (34), we need to find $\left[\lambda I + \mathbf{H}_1^T\mathbf{H}_1 + \cdots + \mathbf{H}_s^T\mathbf{H}_s\right]^{-1}$ according to the Levenberg–Marquardt method. By applying the matrix inverse lemma [Eq. (29)] recursively, this can be done incrementally for each available $\mathbf{H}_i$, $i = 1,2,\ldots,s$. Hence to solve Eq. (34), we need to find each Hessian matrix and the corresponding $(\mathbf{H}_i^T\mathbf{H}_i)^{-1}$, $i = 1,2,\ldots,s$. For notation simplicity, we omit the subscript i hereafter. Since $\mathbf{H}^T\mathbf{H}$ is singular, $(\lambda \mathrm{I} + \mathbf{H}^T\mathbf{H})^{-1}$ is used to replace $(\mathbf{H}^T\mathbf{H})^{-1}$ as mentioned in the last subsection. To reduce the computation load in solving Eq. (34), we use the block Hessian matrix $\mathbf{H}_b$ to approximate $\mathbf{H}$ as discussed in the previous sections.

Consider the block Hessian matrix, $\mathbf{H}_b$, defined by Eqs. (21) and (24), which are repeated here,

$$\mathbf{H}_b = \text{diagonal } \{\mathbf{H}_b^{(1)}, \mathbf{H}_b^{(2)}, \cdots, \mathbf{H}_b^{(L-1)}\} \tag{36}$$

where $\mathbf{H}_b^{(p)} = \sum_{k=1}^{\#(p+1)-1} \boldsymbol{\eta}_k^{(p)} \left[\mathbf{0}_{k-1}, \mathbf{z}^{(p)}, \quad \mathbf{0}_{\#(p+1)-1-k}\right]$, $p = 1, 2, \ldots, L-1$. Due to the block structure of the block Hessian matrix, the inverse of $(\lambda I + \mathbf{H}_b^T\mathbf{H}_b)$ can be obtained by finding the inverse of each block (corresponding to one layer of the MLP), $(\lambda I + (\mathbf{H}_b^{(p)})^T\mathbf{H}_b^{(p)})^{(-1)}$, as we did in Section 3.1. According to Lemma 1, $\mathbf{H}_b^{(p)}$ is a symmetric matrix, i.e. $(\mathbf{H}_{\mathrm{b}}^{(p)})^{\mathrm{T}} = \mathbf{H}_{\mathrm{b}}^{(p)}$, so we have

$$(\mathbf{H}_{\mathrm{b}}^{(p)})^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}^{(p)} = \mathbf{H}_{\mathrm{b}}^{(p)}(\mathbf{H}_{\mathrm{b}}^{(p)})^{\mathrm{T}} = \sum_{k=1}^{\#(p+1)-1} \|\mathbf{z}^{(p)}\|^2 \boldsymbol{\eta}_k^{(p)}(\boldsymbol{\eta}_k^{(p)})^{\mathrm{T}}$$

$$= \|\mathbf{z}^{(p)}\|^2 \cdot \sum_{k=1}^{\#(p+1)-1} \boldsymbol{\eta}_k^{(p)}(\boldsymbol{\eta}_k^{(p)})^{\mathrm{T}} \quad (37)$$

for $p = 1,2,\ldots,L-1$. We can then apply the matrix inverse lemma to find

$$(\lambda I + \mathbf{H}_{\mathrm{b}}^{(p)}(\mathbf{H}_{\mathrm{b}}^{(p)})^{\mathrm{T}})^{-1} = \left(\lambda I + \|\mathbf{z}^{(p)}\|^2 \cdot \sum_{k=1}^{\#(p+1)-1} \boldsymbol{\eta}_k^{(p)}(\boldsymbol{\eta}_k^{(p)})^{\mathrm{T}}\right)^{-1} \quad (38)$$

recursively.

Let us now analyze the computation complexity of the above approach. Let $n_p = (\#(p+1) - B_p)(\#p)$, where $B_p = 1$ if $p \neq L-1$ and $B_p = 0$ if $p = L-1$. For general matrix multiplication, the number of multiplications needed in computing $\mathbf{H}_{\mathrm{b}}^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}$ is $\sum_{i=1}^{L-1} n_i^3$, and the number of additions needed is $\sum_{i=1}^{L-1}(n_i - 1)n_i^2$. However, if we take the singular property of $\mathbf{H}_{\mathrm{b}}^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}$ into account and compute $\mathbf{H}_{\mathrm{b}}^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}$ according to Eq. (37), the number of multiplications needed in computing $\mathbf{H}_{\mathrm{b}}^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}$ is $\sum_{p=1}^{L-1}[(\#(p+1) - B_p)n_p^2 + \#p + n_p]$, and the number of additions needed is $\sum_{p=1}^{L-1}[([(\#(p+1) - B_p)n_p^2 + n_p - 1]$. For example, consider a four-layer MLP with structure N_{2341}^4, where $n_1 = 4$, $n_2 = 9$, $n_3 = 4$. The number of multiplications needed for computing $\mathbf{H}_{\mathrm{b}}^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}$ using normal matrix multiplication is $64 + 729 + 64 = 857$, and the number of additions needed is $48 + 648 + 48 = 744$. For the proposed simplified method, the number of multiplications needed is $(2 + 4 + 16 \times 2) + (3 + 9 + 81 \times 3) + (4 + 4 + 16) = 317$, and the number of additions needed is $35 + 251 + 19 = 305$. Further, if the symmetric property of the $\mathbf{H}_{\mathrm{b}}$ matrix is considered, the computation load analyzed previously can be cut in half.

Again, the singular property of block Hessian matrix simplifies the computation of $\mathbf{H}_{\mathrm{b}}^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}$ greatly in using the LS method for updating network weights. Also, like the analysis in Section 3.1, the matrix inverse lemma reduces the load in computing $(\lambda I + (\mathbf{H}_{\mathrm{b}}^{(p)})^{\mathrm{T}}\mathbf{H}_{\mathrm{b}}^{(p)})^{-1}$ by $n_p/\#p$ times as compared with the normal approach.

4. Simulations

In this section, we shall compare the performance of the proposed second-order learning algorithm in Section 3 to that of the pure gradient descent rule (backpropagation algorithm), Newton's method in which the whole Hessian matrix is computed, and a commonly used simplified Newton's method in which only the diagonal terms of the Hessian matrix are considered. The performance comparison indexes are learning speed and convergence accuracy. Since the MLP is a universal approximator of continuous functions, we shall use three continuously differentiable functions with different input–output dimensions as examples to test the learning speed and convergence accuracy of the proposed second-order learning algorithm.

It is noted that in applying Newton's method to minimize an error function, we simply let $\nabla E = 0$ [see Eq. (32)] and thus the found extreme point could be either a minimum point or a maximum point. To avoid wrong convergent directions in using Newton's method, we propose three protection methods, denoted by PT1, PT2 and PT3, respectively. In the PT1 method, we compare the directions (signs) of the computed $\Delta\mathbf{w}$ and $-\nabla E$. If they are in the same direction, then we adopt the update $\Delta\mathbf{w}$. Otherwise, we update the network weights according to $-\nabla E$. With the PT1 method, the network weights are updated by the general gradient descent rule when $E(\mathbf{w})$ is convex, and are updated by the proposed second-order learning algorithm when $E(\mathbf{w})$ is concave. In the PT2 method, we reverse the sign of the bias term in Eq. (11), and keep the sign of the backpropagation term in Eq. (11) unchanged. This makes $E(\mathbf{w})$ become concave when it is convex to push the weights away from the maximum points. When $E(\mathbf{w})$ is concave and near the region $\nabla E \approx$ zero, since the backpropagation term in Eq. (11) is dominant (this can be proved by order analysis), the sign change of the bias term will not affect the curvature of $E(\mathbf{w})$. The PT3 method is the simplest one; it ignores the bias term in Eq. (11) and leaves only the backpropagation term. In the three protection methods, only PT1 computes the exact block Hessian matrix. However, the (approximated) block Hessian matrices computed in all the three methods own the properties mentioned in the three theorems in Section 2. Especially, these (approximated) block Hessian matrices converge to the same curvature, which is determined by the backpropagation term in Eq. (11), when $\nabla E \approx$ zero.

We shall next use three typical examples concerning single-input-single-output (SISO), multi-input-single-output (MISO), and multiinput-multioutput (MIMO) systems, respectively, to verify the convergence speed and accuracy of the proposed second-order learning algorithm based on the block Hessian matrix under the protections of PT1, PT2 and PT3, respectively. (In the following, we shall call the proposed second-order learning algorithm with the PT1, PT2, and PT3 protection schemes as PT1, PT2, and PT3 methods, respectively, for short.) The performance will be compared with that of the pure gradient descent rule (i.e. backpropagation (BP) algorithm), Newton's method computing the full Hessian matrix ($\mathbf{H}$ method), and the simplified Newton's method computing only the diagonal Hessian matrix ($\mathbf{H}_{\mathrm{d}}$ method). Here, the LS update method proposed in Section 3.2 and the PT2 protection method are also used in the $\mathbf{H}$ method.

Example 1: consider a four-layer MLP with structure $N_{2,3,4,1}$. We fix the weights of this network (denoted by 'teaching MLP'), and use another MLP (denoted by 'training MLP') with the same topology to learn the input–output relationship of the teaching MLP. The training data are

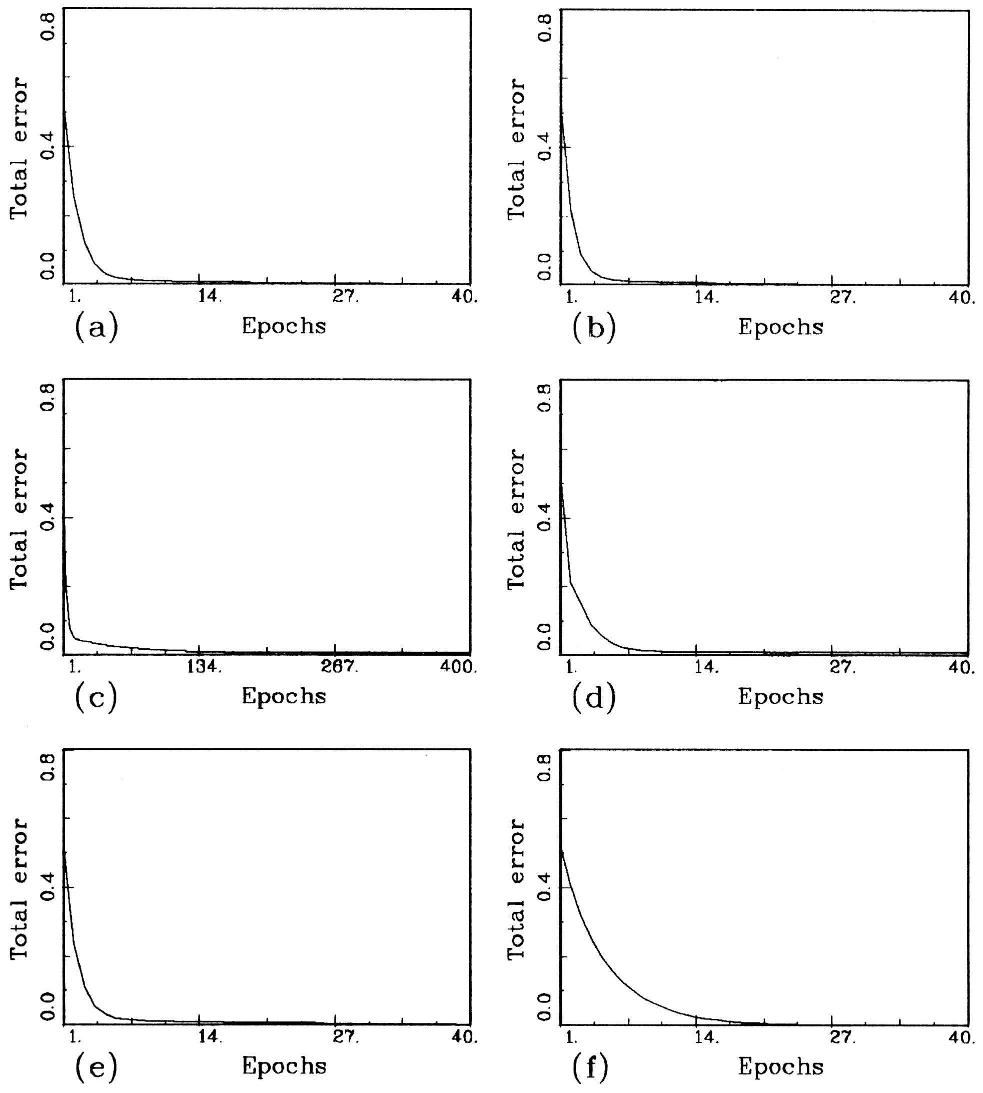


Fig. 3. Convergence curves of various learning algorithms in *Example 1*.

collected by feeding random numbers uniformly distributed in [− 3.0, 3.0] to the teaching MLP and recording its output values. A total of 60 training input–output pairs are collected. The weights of the training MLP are initialized randomly, and then updated by the six first-order and second-order learning algorithms mentioned previously. The learning results are shown in Fig. 3. Fig. 3(a) and (b) show, respectively, the convergence curves (accuracy) of the PT1 and PT2 methods after 40 learning iterations. The convergence curve of the BP algorithm after 400 iterations is shown in Fig. 3(c). It is observed that the convergence accuracy of BP is smaller than that of PT2 by one order, even after ten cycles of the learning iterations, which takes nearly the same actual computation time as PT1 and PT2. Fig. 3(d) shows the learning curve of the $\mathbf{H}_d$ method, which displays poor convergence accuracy. The learning curve of the PT3 method is shown in Fig. 3(e), which is very similar to that of the PT2 method in convergence speed and

Table 2
Range of weight variations and convergence accuracy of the training MLP under six different learning algorithms in *Example 1*, where $\mathbf{w}^{(k)}$ means the weight vectors in layer k, $k = 1,2,3$

Method	$\|\mathbf{w}^{(1)} - \mathbf{w}^{(1)}(0)\|$	$\|\mathbf{w}^{(2)} - \mathbf{w}^{(2)}(0)\|$	$\|\mathbf{w}^{(3)} - \mathbf{w}^{(3)}(0)\|$	Epochs	Error
PT1	0.4046	0.3313	0.8875	40	$2.944 \times 10^{\wedge\{-3}\}$
PT2	0.3332	0.3255	0.7705	40	$1.796 \times 10^{\wedge\{-3}\}$
BP	0.0603	0.0454	0.1325	400	$7.320 \times 10^{\wedge\{-3}\}$
$\mathbf{H}_d$	0.2081	0.2807	0.0908	40	$1.066 \times 10^{\wedge\{-2}\}$
PT3	0.3868	0.2959	0.8507	40	$3.230 \times 10^{\wedge\{-3}\}$
$\mathbf{H}$	0.1571	0.0812	0.2189	40	8.121×10^{-4}

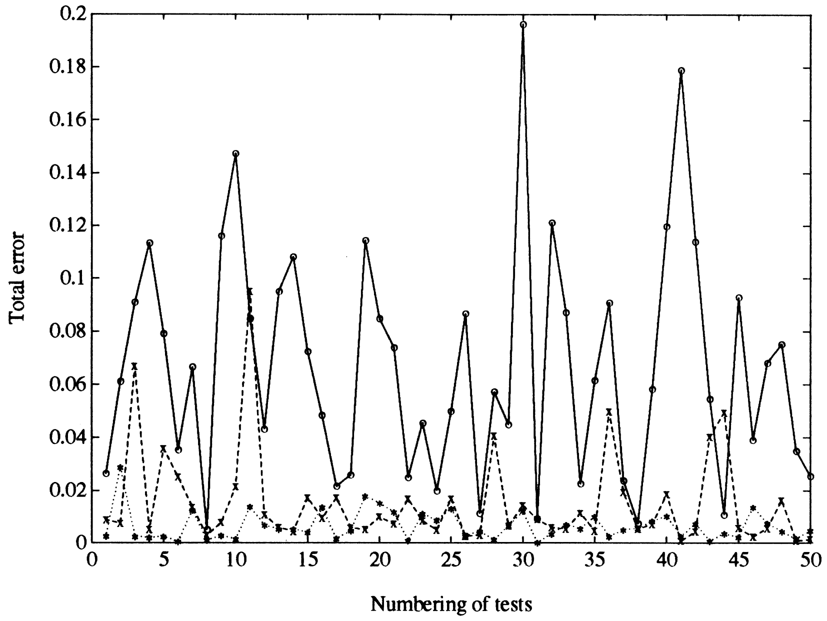


Fig. 4. Repeated tests of convergence accuracy of three learning algorithms: BP (solid line), PT2 (dotted line), and PT3 (broken line) in *Example 1*.

accuracy. The learning curve of the **H** method under the PT2 protection is shown in Fig. 3(f). We observe that the proposed second-order learning algorithm based on the block Hessian matrix can achieve nearly the same convergence accuracy as that of the Newton's method that computes the full Hessian matrix, although the former takes much less computation time per iteration than the latter. Table 2 shows the moving distances of the weight vectors of the training MLP (the distance between the final and initial weight vectors) and convergence accuracy under different learning algorithms. To obtain reliable test results, we repeat the above test fifty times; at each time, the weights of the training MLP are re-initialized randomly. In the repeated tests, we use the BP, PT2, and PT3 methods to train the training MLP, and record the respective total error after 40 (for PT2 and PT3) or 400 (for BP) iterations. The results are shown in Fig. 4. It is found that the mean of total errors for the PT2 method is 0.0064, and the standard deviation is 0.0056; the mean of total errors for the PT3 method is 0.0151, and the standard deviation is 0.0183; the mean of total errors for the BP method is 0.0669, and the standard deviation is 0.0437. These statistics numbers are comparable to those in Table 2 in magnitude order. From Table 2 and Fig. 4, we find that the proposed block-Hessian-matrix-based second-order learning algorithm has much larger search range than the gradient descent method, and preserves high convergence accuracy and speed.

Example 2: consider the Vander Pol's equation, $y(x_1,x_2)=(1/20)(x_1(1-x_2^2)-x_2)$, which is a two-input–single-output system. In this example, we use a MLP, $N_{3,10,5,1}$, with random initial weights to learn this system. The training data are collected by randomly choosing (x_1, x_2) values uniformly distributed in $[-2.5, 2.5]\times[-2.5, 2.5]$. A total of 100 training data are used. The six learning algorithms in *Example 1* are used here again to train the MLP. The results are shown in Fig. 5. Fig. 5(a) and (b) show the learning curves of PT1 and PT2, respectively. It appears that the PT2 methods still keeps high convergence accuracy and speed. The learning curve of BP after 800 iterations is shown in Fig. 5(c). The whole actual learning time of BP is larger than that of PT1 and PT2, but the convergence accuracy of BP is much worse than that of PT1 and PT2. In using the $\mathbf{H}_d$ method, we found that the same λ value in Eq. (28) as that used in the PT1 and PT2 methods could not make the $\mathbf{H}_d$ method converge. Hence we reset the parameter λ from 0.5×10^{-4} to 0.5. The

Table 3
Range of weight variations and convergence accuracy of the training MLP under six different learning algorithms in *Example 2*, where $\mathbf{w}^{(k)}$ means the weight vectors in layer k, $k=1,2,3$

Method	$\|\mathbf{w}^{(1)}-\mathbf{w}^{(1)}(0)\|$	$\|\mathbf{w}^{(2)}-\mathbf{w}^{(2)}(0)\|$	$\|\mathbf{w}^{(3)}-\mathbf{w}^{(3)}(0)\|$	Epochs	Error
PT1	3.7414	32.1940	2.8925	160	$3.922\times10^{\wedge\{-1\}}$
PT2	6.6627	18.9966	3.6575	80	$4.819\times10^{\wedge\{-2\}}$
BP	1.9902	1.7518	2.5195	800	$4.546\times10^{\wedge\{-1\}}$
$\mathbf{H}_d$	0.0939	0.0640	1.2062	80	1.454
PT3	7.5917	21.1058	1.9849	80	$1.230\times10^{\wedge\{-1\}}$
H	4.0045	24.8150	23.047	80	3.058×10^{-2}

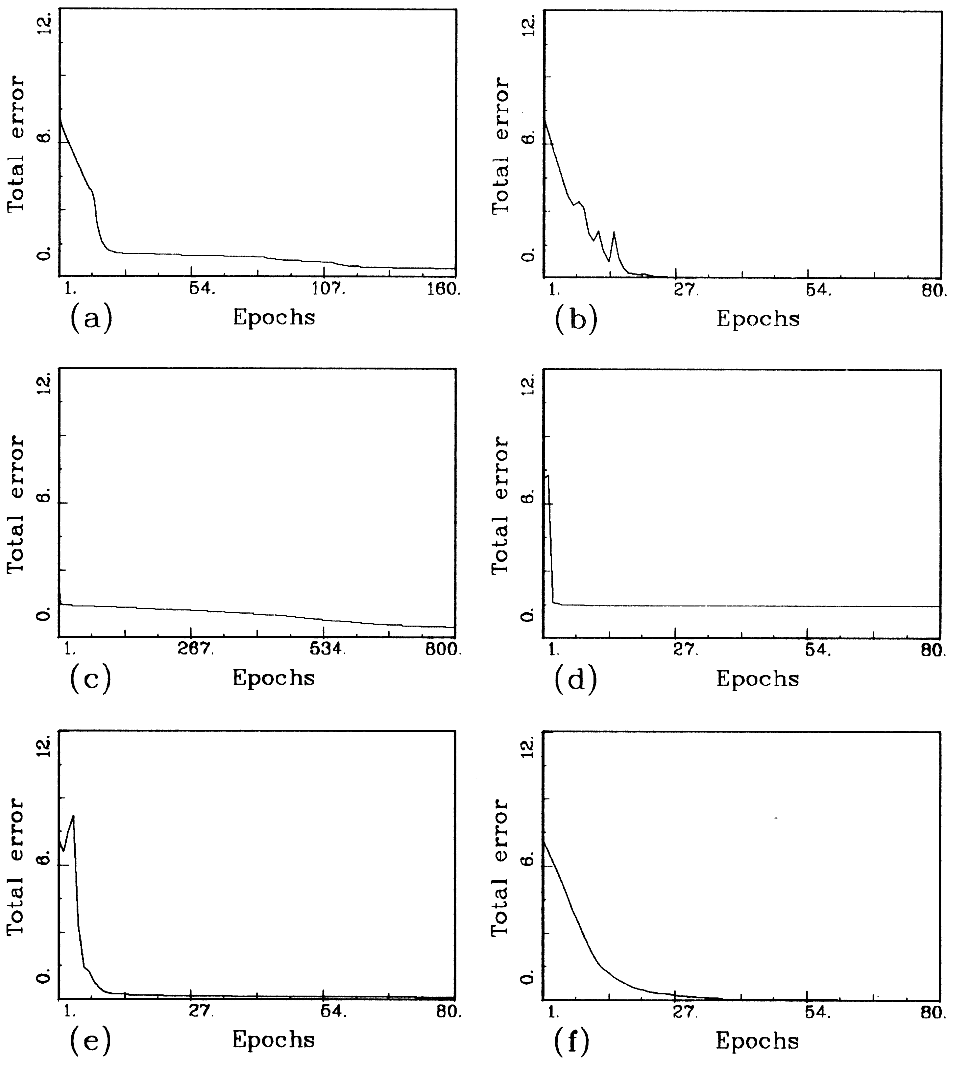


Fig. 5. Convergence curves of various learning algorithms in *Example 2*.

corresponding learning curve is shown in Fig. 5(d), Displaying similar convergence accuracy and speed to the BP method. The learning curves of the PT3 and **H** methods are shown in Fig. 5(e) and (f), respectively, which are both similar to the learning curve of PT2. Table 3 lists the range of weight variations and convergence accuracy under various learning algorithms. Again, the proposed second-order learning algorithm, like the **H** method, shows the largest search range on the weight space.

Example 3: consider a system described by

$$y(x_1,x_2)=\left[\frac{1}{5}\exp\left(\frac{\sin(x_1)-\cos(x_2)}{4+\cos(x_2)}\right),\cos(x_1)\sin(2x_2)\right],$$

which is a two-input–two-output system, where the two inputs are x_1, x_2, and the two outputs are $(1/5)\exp[(\sin(x_1)-\cos(x_2))/(4+\cos(x_2))]$ and $\cos(x_1)$-$\sin(2x_2)$. In this example, we use a $N_{3,7,5,2}$ MLP, which has two output nodes, to learn the behavior of this system. The

Table 4
Range of weight variations and convergence accuracy of the training MLP under six different learning algorithms in *Example 3*, where $\mathbf{w}^{(k)}$ means the weight vectors in layer k, k = 1,2,3

Method	$\|\mathbf{w}^{(1)}-\mathbf{w}^{(1)}(0)\|$	$\|\mathbf{w}^{(2)}-\mathbf{w}^{(2)}(0)\|$	$\|\mathbf{w}^{(3)}-\mathbf{w}^{(3)}(0)\|$	Epochs	Error
PT1	6.0998	18.0398	5.4619	100	$4.863\times 10^{\wedge\{-1\}}$
PT2	3.5607	45.6548	7.3597	80	$1.539\times 10^{\wedge\{-1\}}$
BP	2.3591	1.1166	1.8174	800	$9.141\times 10^{\wedge\{-1\}}$
$\mathbf{H}_d$	0.2883	0.7666	0.6662	80	6.447
PT3	22.4823	49.1420	49.5137	80	$6.979\times 10^{\wedge\{-1\}}$
H	8.1261	12.9116	67.3783	80	3.718×10^{-1}

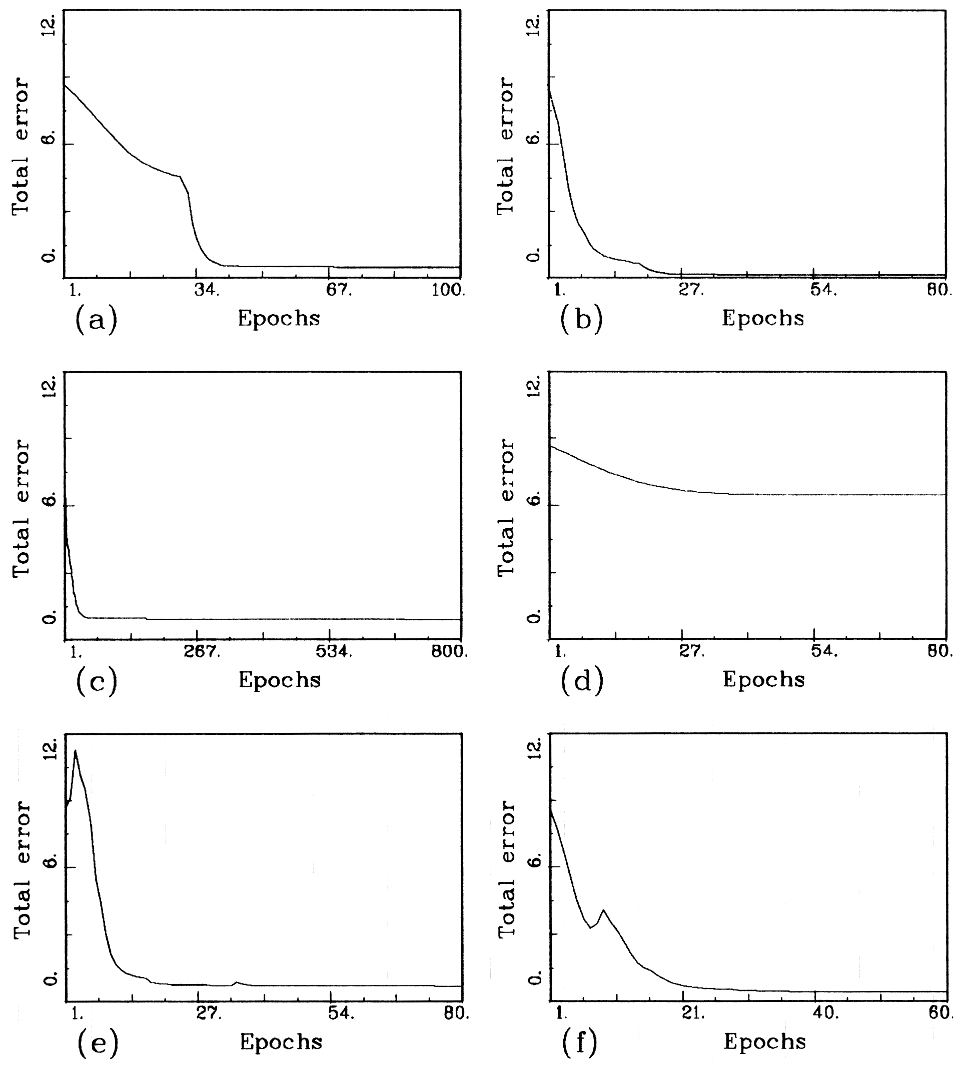


Fig. 6. Convergence curves of various learning algorithms in *Example 3*.

training data are obtained by randomly choosing (x_1, x_2) values uniformly distributed in [− 1,1] × [− 1,1]. A total of 64 training patterns are used. Like the previous two examples, six learning algorithms are used to train the MLP, and the results are shown in Fig. 6. The learning curves of PT1 and PT2 are shown in Fig. 6(a) and (b), respectively. Again PT2 is found to be superior in convergence accuracy and speed. Fig. 6(c) is the learning curve of BP after 800 iterations, which takes longer actual computation time, but achieves much worse accuracy than PT2. The learning curve of the $\mathbf{H}_d$ method, as shown in Fig. 6(d), converges when the parameter λ is reset to be 1.0. The learning curves of the PT3 and **H** methods are shown in Fig. 6(e) and (f), respectively, which are both similar to the learning curve of PT2 again. The ranges of weight variations and convergence accuracy of various learning algorithms are listed in Table 4.

To see the performance of the combination of different learning algorithms, we did the following extra tests. In the first test, we train the MLP using BP until the learning curve converges stably after about 100 epochs, and then apply PT2 on the trained MLP for further training. The corresponding learning curve is given in Fig. 7(a) showing that the PT2 method can further reduce the network output error even after the BP learning. In the second test, we train the MLP using PT2 first until converges and then using BP with the same learning constant as that used for Fig. 6(c). The learning curve of this test is shown in Fig. 7(b), indicating that the learning constant is too large to keep the minimum point located by PT2. The third test is similar to the second one, except that the learning constant of BP is reduced by 100 times. The corresponding learning curve is shown in Fig. 7(c), indicating that BP keeps the minimum point found by PT2 in this case. These tests show that the learning constant affects the convergence accuracy of BP greatly. The aforementioned tests use some hybrid learning schemes, but the PT2 method still show the best performance.

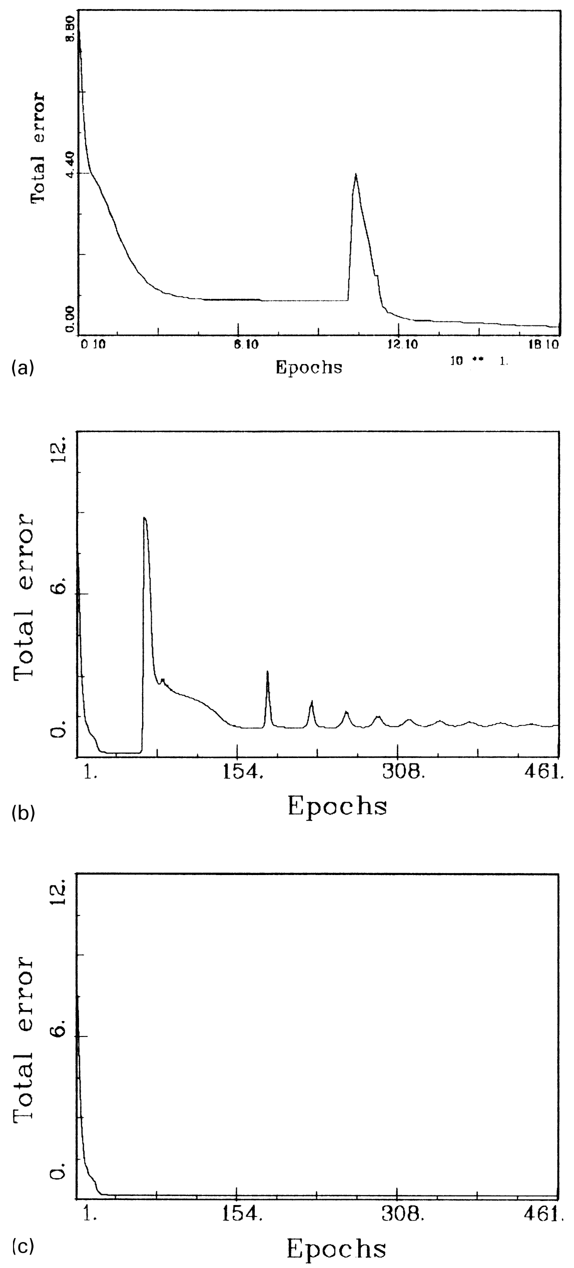


Fig. 7. Convergence curves of hybrid learning algorithms of BP and PT2 in *Example 3*.

It is noted that the MLPs used in the earlier examples all have more than two input nodes and use sigmoidal activation functions, so their Hessian and block Hessian matrices are singular according to Theorems 1 and 2. This substantiates the legal usage of the proposed second-order learning algorithm. The previous simulations lead to the following discussions. Since the proposed block-Hessian-matrix approach is a second-order learning algorithm with quadratic convergence speed, it, unlike the normal back-propagation algorithm, can avoid the phenomena of being trapped in the flat region of error surface. This property generates a higher learning speed and probabilities of locating better local minima than the normal back-propagation algorithm. Simulation results also show that the proposed approach has higher learning speed and convergence accuracy than the diagonal-Hessian-matrix method. This is consistent with our analysis result in Section 2.2 that the diagonal Hessian matrix of a MLP does not own the important properties of the whole Hessian matrix. In other words, the diagonal Hessian matrix of a MLP is an ill approximation of the original Hessian matrix. On the contrary, the block Hessian matrix discussed in this paper keeps the important properties of the original Hessian matrix. Hence, the proposed block-Hessian-matrix approach shows nearly the same convergence accuracy as the one that computes the whole Hessian matrix; however, the former needs much less computation power, and thus has higher learning speed, than the latter. Moreover, the proposed protection methods associated with the block-Hessian-matrix approach can avoid the problem of wrong convergence directions in the normal Hessian-matrix-based learning algorithms.

5. Conclusion

A novel second-order learning algorithm for the MLP has been developed in this article, which represents an attractive alternative to standard backpropagation algorithms. The proposed second-order learning algorithm is a revised Newton's method, in which the computation of the Hessian matrix of the MLP (**H**) and its inverse is the kernel part. Several techniques have been presented in developing this algorithm, including the order-based-derivative approach to deriving formulas for **H**, the network implementation of **H**, a forward–backward propagation scheme for computing **H**, the block Hessian matrix of the MLP to approximate **H**, the use of matrix inverse lemma to find the inverse of block Hessian matrix, the least squares update method for updating the weights of the MLP, and three protection methods to insure correct convergence directions. Further, several lemmas and theorems have been provided to show the important properties of **H**, and verify that the proposed block Hessian matrix is a good approximation of **H**. The theoretic analysis not only uncovers the nature of the Hessian matrix of the MLP, but also provides important insight into developing an efficient second-order learning algorithm for the MLP. The proposed algorithm has solved the major drawbacks of the standard backpropagation algorithm and of normal Newton's method in training the MLP. Extensive computer simulations and performance comparisons with backpropagation algorithm and other revised Newton's methods have demonstrated the fast learning speed and high convergence accuracy of the proposed learning algorithm. Formal mathematic analysis on the convergence speed and accuracy of the proposed second-order learning algorithm protected by the proposed protection methods will be the subject of our future work.

References

Battiti, R. (1992). First and second order methods for learning between steepest descent and Newton's method. *Neural Computation*, *4*, 141–166.

Becker, S., & LeCun, Y. (1989). Improving the convergence of back-propagation learning with second order methods. In D. S. Touretzky, G. E. Hinton, & T. J. Sejnowski (Eds.), *Proc. 1988 Connectionist Models Summer School*. San Mateo, CA: Morgan Kaufmann, pp. 29–37.

Bello, M. G. (1992). Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks. *IEEE Transactions on Neural Networks*, *3*, 864–875.

Goodwin, G. C., & Sin, K. S. (1984). *Adaptive Filtering Prediction and Control* (chap. 3). Englewood Cliffs, NJ: Prentice-Hall.

Haykin, S. (1994). *Neural Networks* (chap. 7). New York: Macmillan.

Kailath, T. (1980). *Linear Systems*. Englewood Cliffs, NJ: Prentice-Hall.

LeCun, Y. (1989). Generalization and network design strategies. In *Connectionism in Perspective*. Amsterdam: North Holland, 143–155.

Lin, C. T., & Lee, C. S. G. (1996). *Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hall.

Luenberger, D. G. (1984). *Linear and Nonlinear Programming.* Reading, MA: Addision-Wesley.

Makram-Ebeid, S., Sirat, J. A., & Viala, J. R. (1989). A rationalized back-propagation learning algorithm. In *Proc. Int. Joint Conf. Neural Networks*. Washington, DC, pp. 373–380. NJ: IEEE.

Piche, S. W. (1994). Steepest descent algorithms for neural network controllers and filters. *IEEE Transactions on Neural Networks*, *5*, 198–212.

Ricotti, L. P., Ragazzini, S., & Martinelli, G. (1988). Learning of word stress in a sub-optimal second order back-propagation neural network. In *Proc. IEEE Int. Conf. Neural Networks*. San Diego, CA, pp. 355–361. NJ: IEEE.

Silva, F., & Almeida, L. (1990). Acceleration techniques for the backpropagation algorithm. *Lecture Notes in Computer Science*, Vol. 412. Berlin: Springer, pp. 110–119.

Stefanos, K., & Anastassioy, D. (1988). Adaptive training of multilayer neural networks using a least-squares estimation technique. In *Proc. IEEE Int. Conf. Neural Networks.* San Diego, CA, pp. 383–390. NJ: IEEE.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. of the IEEE*, *78*, 1550–1560.