Available online at www.sciencedirect.com
ScienceDirect
NEUROCOMPUTING
Neurocomputing 70 (2006) 544–558
www.elsevier.com/locate/neucom

A new training and pruning algorithm based on node dependence and Jacobian rank deficiency[☆]

Jinhua Xu[a], Daniel W.C. Ho[b,*]

[a]Department of Computer Science, East China Normal University, Shanghai 200062, PR China
[b]Department of Mathematics, City University of Hong Kong, Kowloon, Hong Kong, PR China

Received 24 February 2005; received in revised form 5 November 2005; accepted 9 November 2005
Available online 3 March 2006
Communicated by J. Zhang

Abstract

In this paper, a new subset-based training and pruning (SBTP) algorithm is proposed based on the relationship between node dependence and Jacobian rank deficiency. At each training iteration, the orthogonal factorization with column permutation is applied to the output of the nodes in the same hidden layer to identify the dependent nodes. The output weights of the dependent nodes will be set as zeros, and the output weights of the independent nodes will be recalculated to maintain the original input–output behavior. Then, only the weights of the independent nodes will be trained using the Levenberg–Marquardt (LM) algorithm at this iteration, while keeping the weights of the dependent nodes unchanged. In this way, the computational cost of the LM algorithm will be reduced significantly. After the training process, a unit-based optimal brain surgeon (UB-OBS) pruning method is used to prune the insensitive hidden units to further reduce the size of the neural network, and no retraining is needed. Simulations are presented to demonstrate the effectiveness of the proposed approach.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Neural networks; Pruning; Jacobian matrix; Rank deficiency; Generalization

1. Introduction

The feedforward neural network (NN) is one of the most popular NN topologies. One of the difficulties of using the feedforward NN is to determine the network architecture. In general, when a network is too small, it may not learn the problem well. However, a network which is too large may lead to overfitting and poor generalization performance. Algorithms that can find an appropriate network architecture automatically are thus highly desirable. There are three major approaches to tackle this problem. The first involves using a larger than needed network and training it until an acceptable solution is found. After this, some hidden units or weights are removed if they are no longer actively used. Methods using this approach are called pruning algorithms [25]. The second approach, which corresponds to constructive algorithms, starts with a small network and then grows additional hidden units and weights until a satisfactory solution is found [1,9,14,19,24,33]. The third approach is regularization, which involves the addition of a penalty term to the objective function to be minimized [11,15,27]. While the regularization models are generally easy to implement, the value of the regularization parameter may present problems. Regularization therefore requires a delicate balance between the normal error term and the penalty term. Another disadvantage of penalty terms is that they tend to create additional local minima, increasing the possibility of converging to a bad local minimum [8]. Some research has been done on combining the constructive algorithm and the pruning algorithms to determine the NN architecture automatically [6,16,23]. In [6], a forward backward and model selection algorithm for constructing a hybrid regression network of radial and perceptron hidden units is introduced. The algorithm determines the unit type to be added and the

[☆]This work is supported by Hong Kong CERG CityU 101004.
*Corresponding author.
E-mail addresses: jhxu@cs.ecnu.edu.cn (J. Xu), madaniel@cityu.edu.hk (D.W.C. Ho).

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.11.005

number of hidden nodes, then unnecessary weights are pruned using model selection criteria. In [16], a cascade NN design algorithm is described for designing compact two-hidden-layer NNs.

Recent interest has been growing on pruning algorithms that start with an oversized network and remove unnecessary network parameters, either during training or after convergence to a local minimum. Network parameters that are considered for removal are individual weights, hidden units and input units. Sensitivity analysis has been used successfully to prune irrelevant parameters from feedforward NNs. Sensitivity analysis techniques quantify the relevance of a network parameter as the influence that small parameter perturbations have on a performance function. There are two main approaches to NN sensitivity analysis, where the difference is in the performance function used, which can be either the objective function [4,5,13,21,22,28,32,31] or the NN output function [26,38]. In [8], it is shown that objective function and output function sensitivity analysis are conceptually the same.

The sensitivity measures could fail to identify possible correlations among nodes [4,13,21,22]. Other post-training pruning algorithms based on the correlations among nodes are proposed in [2,3,10,17,18,29,30,37]. In [2], the pruning algorithm is formulated in terms of solving a system of linear equations, and a conjugate gradient algorithm is used for solving it. In [3], a hidden unit pruning algorithm called linear dependence pruning utilizing sets of linear equations is presented. In [10], Fletcher et al. use the singular value decomposition (SVD) of the conditional Fisher information matrix, together with likelihood-ratio tests to determine irrelevant hidden units. In [17], synchronous pairs or contrary pairs of hidden units are chosen by evaluating a firing similarity, and then fused so that features accumulated in both units are preserved as possible. In [18], SVD along with orthogonal (QR) factorization with column pivoting is used forsubset selection to optimize the size of feedforward NNs. The reduced-size network is reinitialized and retrained to the desired error level.

One of the disadvantages of post-training pruning algorithms is that the computational cost is heavy since the majority of the training time is spent on networks larger than necessary [19]. In [37], a subset-based training and pruning (SBTP) method is proposed based on Jacobian rank deficiency, and an extra penalty term on weights is added to the objective function corresponding to weight pruning, hence it belongs to the regularization approach. Since only a subset of weights is trained during the training process, the computational cost of the training algorithm is reduced significantly. However, the redundancy of weights cannot be estimated by using Jacobian rank deficiency. Therefore, the redundant weights cannot be removed directly according to the rank deficiency. The convergence of the algorithm cannot be guaranteed and some extra tuning parameters are required in the training and pruning process. The improvement on these shortcomings are shown in Section 3.

The phenomenon of premature saturation (PS) of the network output units when the units are mapped by sigmoid-like functions has been widely recognized by researchers and NN users [34]. This undesirable phenomenon, sometimes referred to in the literature as the flat spot problem, is characterized by the temporary trapping of the network output units at saturated activation levels during the early stage of the training process. The slow convergence problem of backpropagation (BP) training posed by PS of the network output units has been addressed in some works [34], but, to our knowledge, no work has been done to identify and prune the redundant nodes based on the premature problem during the training process.

In this paper, a new SBTP algorithm is proposed based on the relationship between node dependence and Jacobian rank deficiency. At each training iteration, the OR factorization with column permutation is applied to the output of the nodes in the same hidden layer to identify the dependent nodes. Then, at this iteration, only the weights of the independent nodes will be trained using the Levenberg–Marquardt (LM) algorithm, while keeping the weights of the dependent nodes unchanged. In this way, the computational cost of the LM algorithm will be reduced significantly. After the training process, a unit-based optimal brain surgeon (UB-OBS) pruning algorithm is used to prune the insensitive hidden units to further reduce the size of the NN, and no retraining is needed. In contrast with the existing post-training pruning algorithms, the proposed training and pruning algorithm has three advantages. First, only a subset of the hidden nodes (independent nodes) are trained at each iteration, and so the excessive training computational cost of the post-training algorithm is avoided. Second, no extra term is added to the objective function, so our algorithm does not use any regularization parameter, and this avoids an unnecessary lengthy tuning phase. Third, no retraining is needed after the post-training pruning.

Section 2 reviews the standard training algorithms for feedforward NNs. Section 3 presents the new SBTP algorithm. In Section 4, a UB-OBS pruning algorithm is described to prune the insensitive hidden units to further reduce the size of the NN. Simulation results are reported in Section 5 and the proposed SBTP algorithm is compared with standard algorithms.

2. Problem formulation

The NNs considered in this paper are feedforward NNs. For simplicity, the algorithm is derived for feedforward NNs with one hidden layer. The activation function of the hidden layer and output layer are sigmoid and linear function, respectively. It is straightforward to extend the algorithm to multiple hidden layer networks.

For one-hidden-layer feedforward network, the input–output relationship can be represented as

$$\hat{\mathbf{y}} = \mathbf{W}^2\sigma(\mathbf{W}^1\mathbf{x} + \mathbf{B}^1) + \mathbf{B}^2, \tag{1}$$

where $\mathbf{x} \in R^n, \mathbf{y} \in R^p$ are the network input and output, respectively. n is the input dimension and p is the output dimension. $\mathbf{W}^1 \in R^{m\times n}$ is the weight matrix from the input layer to the hidden layer, and $\mathbf{B}^1 \in R^m$ is the bias vector of the hidden neurons, where m is the number of the hidden neurons. $\mathbf{W}^2 \in R^{p\times m}$ is the weight matrix from the hidden layer to the output layer, and $\mathbf{B}^2 \in R^p$ is the bias vector of the output neurons. In this paper, $\mathbf{B}^2$ is regarded as the weights from a bias unit with a constant output 1 to the output layer, and σ is a differentiable activation function, such as sigmoid functions.

Assume that the hidden neurons in the network are numbered from 1 to m. All the weight parameters related to the ith hidden neurons including all the incoming and outgoing connections and the bias are grouped in a S_i dimensional vector $\bar{\boldsymbol{\theta}}_i$, $i = 1,\ldots,m$. Here

$$\bar{\theta}_i = [w^1_{i1},\ldots,w^1_{in}, b^1_i, w^2_{1i},\ldots,w^2_{pi}]^{\mathrm{T}}.$$

All the biases in the output layer are grouped in $\bar{\boldsymbol{\theta}}_{m+1}$, that is

$$\bar{\boldsymbol{\theta}}_{m+1} = [b^2_1, b^2_2,\ldots,b^2_p]^{\mathrm{T}}.$$

Then $\boldsymbol{\theta}$ can then be represented as

$$\boldsymbol{\theta} = [\bar{\theta}^{\mathrm{T}}_1,\ldots,\bar{\theta}^{\mathrm{T}}_m,\bar{\theta}^{\mathrm{T}}_{m+1}]^{\mathrm{T}} \in R^S, \tag{2}$$

where $S = m*(n+1+p)+p$ is the number of the weight parameters of the network.

With batch learning, a set of training patterns $\{\mathbf{x}(t),\mathbf{y}(t), t = 1,\ldots,N\}$ is given. Here $\mathbf{x}(t) \in R^n$ and $\mathbf{y}(t) \in R^p$ are the input and desired output of the network, respectively. The output of the network is

$$\hat{\mathbf{y}}(t,\theta) = \mathbf{W}^2\sigma(\mathbf{W}^1x(t)+\mathbf{B}^1)+\mathbf{B}^2. \tag{3}$$

If the error is defined as

$$\mathbf{e}(t):=\hat{\mathbf{y}}(t,\theta)-\mathbf{y}(t), \tag{4}$$

then the training problem can be formulated as an optimization problem as follows:

$$\min_\theta V(\theta) = \frac{1}{2}\sum_{t=1}^{N}\mathbf{e}^{\mathrm{T}}(t)\mathbf{e}(t) = \frac{1}{2}\sum_{t=1}^{N}\sum_{i=1}^{p}e^2_i(t) = \frac{1}{2}E^{\mathrm{T}}E, \tag{5}$$

where $E^{\mathrm{T}} = [\mathbf{e}^{\mathrm{T}}(1),\mathbf{e}^{\mathrm{T}}(2),\ldots,\mathbf{e}^{\mathrm{T}}(N)]$. Let the Jacobian matrix be

$$J(\theta) = \frac{\partial E}{\partial\theta} = \begin{bmatrix}\tilde{J}(1)\\ \tilde{J}(2)\\ \vdots\\ \tilde{J}(N)\end{bmatrix} = \begin{bmatrix}\dfrac{\partial\mathbf{e}(1)}{\partial\theta}\\ \dfrac{\partial\mathbf{e}(2)}{\partial\theta}\\ \vdots\\ \dfrac{\partial\mathbf{e}(N)}{\partial\theta}\end{bmatrix}, \tag{6}$$

$$\tilde{J}(i) = \frac{\partial\mathbf{e}(i)}{\partial\theta} = \begin{bmatrix}\dfrac{\partial e_1(i)}{\partial\theta_1} & \dfrac{\partial e_1(i)}{\partial\theta_2} & \cdots & \dfrac{\partial e_1(i)}{\partial\theta_S}\\ \dfrac{\partial e_2(i)}{\partial\theta_1} & \dfrac{\partial e_2(i)}{\partial\theta_2} & \cdots & \dfrac{\partial e_2(i)}{\partial\theta_S}\\ \vdots & \vdots & \vdots & \vdots\\ \dfrac{\partial e_p(i)}{\partial\theta_1} & \dfrac{\partial e_p(i)}{\partial\theta_2} & \cdots & \dfrac{\partial e_p(i)}{\partial\theta_S}\end{bmatrix}, \tag{7}$$

where

$$\frac{\partial e_i(t)}{\partial w^1_{i',j'}} = w^2_{i,i'}\sigma'_{i'}(t)x_{j'}(t), \quad i' = 1,\ldots,m,\ j' = 1,\ldots,n, \tag{8}$$

$$\frac{\partial e_i(t)}{\partial b^1_{i'}} = w^2_{i,i'}\sigma'_{i'}(t), \quad i' = 1,\ldots,m, \tag{9}$$

$$\frac{\partial e_i(t)}{\partial b^2_{i'}} = \delta_{i,i'}, \quad i' = 1,\ldots,p, \tag{10}$$

$$\frac{\partial e_i(t)}{\partial w^2_{i',j'}} = \delta_{i,i'}\sigma_{j'}(t), \quad i' = 1,\ldots,p,\ j' = 1,\ldots,m. \tag{11}$$

Here $\sigma_i(t) = \sigma(W^1_ix(t)+b^1_i)$ is the output of the ith hidden node for the tth training pattern, W^1_i is the ith row of W^1, and σ' is the derivative of σ.

$$\delta_{i,i'} = \begin{cases}1, & i = i',\\ 0, & i \neq i'.\end{cases} \tag{12}$$

The BP algorithm is a gradient descent method, which can be described as follows:

$$\theta(k+1) = \theta(k) - \eta J^{\mathrm{T}}E, \tag{13}$$

where $\eta > 0$ is the learning rate. The BP algorithm has been widely used for training multilayer perceptron (MLP) networks. However, several drawbacks of the BP algorithm have been observed [35]: (1) its convergence speed is usually too low, its convergence accuracy is hard to control; (2) it is easily stuck in bad local minimum. It is known that Newton's method and its modified versions can be applied to overcome these drawbacks.

The Hessian matrix is defined as

$$H = \frac{\partial^2V}{\partial\theta^2} = \frac{\partial}{\partial\theta}(J^{\mathrm{T}}E) = J^{\mathrm{T}}J + \frac{\partial^2E^{\mathrm{T}}}{\partial\theta^2}E. \tag{14}$$

Neglecting the second term, the Hessian matrix is approximated by the first term

$$H \approx J^{\mathrm{T}}J = \sum_{i=1}^{N}\tilde{J}^{\mathrm{T}}(i)\tilde{J}(i). \tag{15}$$

The Gauss–Newton method can be obtained by solving

$$H\delta\theta = -J^{\mathrm{T}}E. \tag{16}$$

If $J^{\mathrm{T}}J$ is invertible, then the Gauss–Newton algorithm can be obtained as follows:

$$\theta(k+1) = \theta(k) - H^{-1}J^{\mathrm{T}}E. \tag{17}$$

Since J^TJ may not be invertible, the LM algorithm [12] is shown as

$$\theta(k+1) = \theta(k) - (H + \mu I)^{-1} J^T E, \tag{18}$$

where μ is a positive number, which can be adjusted appropriately. Whenever an update results in an increased $V(\theta)$, then the update will be neglected and μ is multiplied by some factor $\beta(>1)$. When an update reduces $V(\theta)$, μ is divided by β. Compared with Eqs. (13) and (17), it can be seen that when μ is large, the algorithm becomes gradient descent (BP); while for small μ, the algorithm becomes Gauss–Newton. The LM algorithm can be considered a trust-region modification to the Gauss–Newton algorithm [12].

The LM algorithm outperforms the basic BP and BP's variations with variable learning rate significantly in terms of training accuracy, convergence properties and overall training time. But one disadvantage of the LM algorithm is that the computation and memory requirements are high within each iteration [36].

The first goal of this paper is to reduce the computational cost of the LM algorithm by subset training. The second goal is to identify and prune the redundant nodes to optimize the size of the NN. Notice that the rank deficiency of the Jacobian matrix in Eq. (6) plays an important role in the implementation of this algorithm. Rank deficiency of the Jacobian matrix on one hand makes the Gauss–Newton algorithm not applicable, since the Hessian matrix is not invertible, but on the other hand it may indicate that some hidden units in the network are dependent. This will be discussed in the next section.

3. Subset-based training and pruning algorithm

In this section, the relationship between node dependence and Jacobian rank deficiency is analyzed, and a new SBTP algorithm is proposed. The significance of this approach is that only the independent nodes will be trained in the training stage, and hence the computational cost of the LM training algorithm will be greatly reduced.

First, the method to identify the dependent nodes will be shown as follows:

At iteration k, the output of the ith hidden node for the tth training pattern is denoted as $\sigma_i(t) = \sigma(W_i^1 x(t) + b_i^1)$. Define the matrix

$$\Phi(k) := [\phi_1, \phi_2, \ldots, \phi_m, \phi_{m+1}], \tag{19}$$

where $\phi_i = [\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(N)]^T$, $i = 1, \ldots, m$, and $\phi_{m+1}(j) = 1$ for $j = 1, \ldots, N$ is the output of a bias unit for all training patterns.

The output of the network at iteration k is denoted as

$$\hat{Y}(k) := [\hat{y}(1), \hat{y}(2), \ldots, \hat{y}(N)]^T$$

and let $W := [W^2, B^2]$ to obtain

$$\Phi(k) W^T = \hat{Y}(k). \tag{20}$$

This means that the output in $\hat{Y}(K)$ is a linear combination of the nodes in $\Phi(k)$. Next, QR factorization with column pivoting is applied to matrix $\Phi(k)$ to detect the independent columns, that is

$$\Phi(k) \cdot \mathbf{P} = \mathbf{Q} \cdot \mathbf{A}, \tag{21}$$

where $\mathbf{Q}$ is a $N * N$ unitary matrix, $\mathbf{A}$ is an upper triangular matrix of the same dimensions as Φ, $\mathbf{P}$ is a permutation matrix with $\mathbf{P}^T\mathbf{P} = \mathbf{P}\mathbf{P}^T = \mathbf{I}$, which is chosen to make the absolute value of the diagonal elements of $\mathbf{A}$ in decreasing order, that is,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m+1} \\ 0 & a_{22} & \cdots & a_{2m+1} \\ & & \cdots & \\ & & a_{rr} & \cdots \\ & & & \cdots \end{bmatrix}. \tag{22}$$

If

$$|a_{rr}|/|a_{11}| < \rho, \tag{23}$$

where ρ is a very small positive number, then $|a_{jj}|/|a_{11}| < \rho$ for $j = r+1, \ldots, \min(m+1, N)$, this means there are r nonzero diagonal elements in A, and therefore there are r independent columns in matrix $\Phi(k)$.

Decompose $\mathbf{A}$ in appropriate block matrix forms as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & 0 \end{bmatrix}, \tag{24}$$

where $\mathbf{A}_{11}$ is an $r * r$ upper triangular matrix, and r is the number of the nonzero diagonal elements in $\mathbf{A}$.

Substitute Eq. (21) into (20), let $\bar{\mathbf{W}} := \mathbf{W}\mathbf{P}$, and it implies that

$$\mathbf{A}\bar{\mathbf{W}}^T = \mathbf{Q}^T \hat{\mathbf{Y}}(k). \tag{25}$$

Correspondingly, decompose $\mathbf{Q}, \bar{\mathbf{W}}$ in appropriate block matrix forms as

$$\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2], \quad \bar{\mathbf{W}} = [\mathbf{W}_r, \mathbf{W}_p], \tag{26}$$

where $\mathbf{W}_r$ and $\mathbf{W}_p$ are $p \times r$ and $p \times (m+1-r)$ matrices, representing the output weight of independent nodes and dependent nodes, respectively.

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{W}_r^T \\ \mathbf{W}_p^T \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^T \\ \mathbf{Q}_2^T \end{bmatrix} \hat{\mathbf{Y}}(k). \tag{27}$$

One can verify that $\mathbf{Q}_2^T \hat{\mathbf{Y}}(k) = 0$. Setting $\mathbf{W}_p = 0$, the independent weight $\mathbf{W}_r$ can be easily obtained by solving the following linear equation:

$$\mathbf{A}_{11}(\mathbf{W}_r)^T = \mathbf{Q}_1^T \hat{\mathbf{Y}}(k). \tag{28}$$

From the above analysis, the output weights of the $m+1-r$ dependent nodes can be set as zeros and the output weights of the independent nodes can be recalculated using Eq. (28) without any effect on the network's input–output behavior.

Next the relationship between node dependence and Jacobian rank deficiency is investigated.

Denote the Jacobian matrix in Eq. (6) as

$$J(\theta) = [\bar{J}_1, \bar{J}_2, \ldots, \bar{J}_{m+1}], \tag{29}$$

where

$$\bar{J}_i = \frac{\partial E}{\partial \bar{\theta}_i} = \left[\frac{\partial E}{\partial w^1_{i1}}, \ldots, \frac{\partial E}{\partial w^1_{in}}, \frac{\partial E}{\partial b^1_i}, \frac{\partial E}{\partial w^2_{1i}}, \ldots, \frac{\partial E}{\partial w^2_{pi}}\right], \tag{30}$$

for $i = 1, \ldots, m$ and

$$\bar{J}_{m+1} = \frac{\partial E}{\partial \bar{\theta}_{m+1}} = \left[\frac{\partial E}{\partial b^2_1}, \ldots, \frac{\partial E}{\partial b^2_p}\right]. \tag{31}$$

Two different cases are investigated as follows:

Case 1. Constant-output node. Assume the ith node is a constant-output node, that is $\sigma_i(t) = \alpha$, $\forall t$, where α is a constant. This means that the ith node is in a saturation region for all training patterns, and then $\sigma'_i(t) = 0$, $\forall t$. From Eqs. (8)–(9), it is shown that

$$\frac{\partial E}{\partial w^1_{ij}} = 0 \quad \text{for } j = 1, \ldots, n \text{ and } \frac{\partial E}{\partial b^1_i} = 0.$$

From Eqs. (10)–(11), it can be deduced that

$$\frac{\partial E}{\partial w^2_{ji}} = \alpha \frac{\partial E}{\partial b^2_j} \quad \text{for } j = 1, \ldots, p.$$

Therefore, all the $n+1+p$ columns of $\bar{\mathbf{J}}_i$ are dependent on other columns in $\mathbf{J}$, and so the rank of $\mathbf{J}$ will be decreased by $n+1+p$.

Case 2. Paralleled or anti-paralleled nodes. Assume the ith node is paralleled or anti-paralleled with jth node. Then $\sigma_i(t) = \alpha\sigma_j(t)$, $\forall t$, where α is a real constant. Assume that the output weights of the ith node are set as zeros, that is, $w^2_{k,i} = 0$, for $k = 1, \ldots, p$. From Eqs. (8)–(9)

$$\frac{\partial E}{\partial w^1_{ik}} = 0, \quad i = 1, \ldots, n \text{ and } \frac{\partial E}{\partial b^1_i} = 0.$$

From Eq. (11), it is shown that

$$\frac{\partial E}{\partial w^2_{ki}} = \alpha \frac{\partial E}{\partial w^2_{kj}} \quad \text{for } k = 1, \ldots, p.$$

Therefore, the $n+1+p$ columns of $\bar{\mathbf{J}}_i$ are dependent on other columns in $\mathbf{J}$, and so the rank of $\mathbf{J}$ will be decreased by $n+1+p$.

Generally, if the ith node is a dependent node, that is, ϕ_i is dependent on other columns in Φ, i.e., $\phi_i = \sum_{j=1}^{n_i} \alpha_j\phi_j$, $j \in [1, 2, \ldots, m+1]\backslash\{i\}, \alpha_j \neq 0$, then all the $n+1+p$ columns of $\bar{\mathbf{J}}_i$ will be dependent on other columns in $\mathbf{J}$.

Therefore, the dependent nodes identified by QR factorization have a direct relationship with the rank deficiency of the Jacobian matrix.

Rearrange the columns of J as follows:

$$\mathbf{J} = [\mathbf{J}_r, \mathbf{J}_p],$$

where $\mathbf{J}_r = [\bar{J}_{l_1}, \bar{J}_{l_2}, \ldots, \bar{J}_{l_r}]$ is the Jacobian matrix of the independent nodes and $J_p = [\bar{J}_{l_{r+1}}, \ldots, \bar{J}_{l_{m+1}}]$ is the Jacobian matrix of the dependent nodes. Since the columns of $\mathbf{J}_p$ are dependent on those in $\mathbf{J}_r$, it can be deduced that

$$\mathbf{J}_p = \mathbf{J}_r \cdot \mathbf{L}, \tag{32}$$

for some appropriate size matrix L with nonzero columns. Correspondingly, the weight vector $\boldsymbol{\theta}$ can be rearranged as follows:

$$\boldsymbol{\theta} = [\theta_r, \theta_p] = [\bar{\theta}_{l_1}, \ldots, \bar{\theta}_{l_r}, \bar{\theta}_{l_{r+1}}, \ldots, \bar{\theta}_{l_{m+1}}].$$

Then Eq. (16) can be decomposed as

$$(J_r)^{\mathrm{T}} J_r \delta\theta_r + (J_r)^{\mathrm{T}} J_r L \delta\theta_p = -(J_r)^{\mathrm{T}} E, \tag{33}$$

$$L^{\mathrm{T}}(J_r)^{\mathrm{T}} J_r \delta\theta_r + L^{\mathrm{T}}(J_r)^{\mathrm{T}} J_r L \delta\theta_p = -L^{\mathrm{T}}(J_r)^{\mathrm{T}} E. \tag{34}$$

The second system equation in (34) can be obtained by multiplying L^{T} on both sides of the first system equation in (33). Therefore, Eq. (34) is dependent on Eq. (33). If $\delta\theta_p = 0$ is chosen, that means the weight parameters related to the dependent nodes remain unchanged or frozen at this iteration, and then from Eq. (33),

$$(J_r)^{\mathrm{T}} J_r \delta\theta_r = -(J_r)^{\mathrm{T}} E. \tag{35}$$

Therefore, the weights of the independent nodes can be updated using the LM algorithm as follows:

$$\theta_r(k+1) = \theta_r(k) - ((J_r)^{\mathrm{T}} J_r + \mu I)^{-1} (J_r)^{\mathrm{T}} E. \tag{36}$$

Since $(J_r)^{\mathrm{T}} J_r$ may still be not invertible, the term μI is added to avoid such ill conditioning, where μ is a small positive scaler.

Notice that at each iteration, only the Jacobian matrix of the independent nodes J_r requires to be calculated. J_p and L are only there for analysis.

Now the computational cost of our proposed SBTP algorithm is compared with the LM algorithm. For the LM algorithm, at each iteration, the number of weight parameters to be trained is $S = m(n+p+1)+p$. For our training and pruning algorithm, only the weights of r $(<m)$ independent nodes need to be trained, that is, the number of weight parameters to be trained is $S_r = r\,(n+p+1) < S$. Besides, QR factorization in Eq. (21) and output weights recalculation in Eq. (28) are needed in our algorithm (SBTP). The number of multiplications for each part of the calculation is shown in Table 1. Since the number of independent nodes (r) is much less than that of the hidden nodes (m), which will be demonstrated in the simulations later, the computational cost will be decreased significantly.

The differences between our algorithm and the one in [37] are as follows: First, QR factorization is applied to the output of the hidden nodes to detect the dependent nodes in our algorithm; in [37], QR factorization is applied to the Jacobian matrix to detect the redundant weights. Second, since there is a direct relationship between node dependence and Jacobian rank deficiency, no tuning parameter is needed in our algorithm; in [37], the redundancy of weights cannot be estimated by using the Jacobian rank deficiency. Therefore, the weight cannot be removed directly according to the rank deficiency. An extra penalty term on weights is added to the objective function corresponding to

Table 1
Comparison of computational cost for each iteration ($r<m, S_r<S$)

# of multiplication operations	LM	SBTP
J	$2mnpN$	$2rnpN$
$J^{\mathrm{T}}J$	$pNS^2/2$	$pN(S_r)^2/2$
H^{-1}[a]	$S^3/2+3S^2/4+7S/3$	$(S_r)^3/2+3(S_r)^2/4+7S_r/3$
QR[a]	0	$N(m+1)^2/2-(m+1)^3/3$
Output weight[a]	0	$2r^3/3$

[a]The number of multiplication operations is obtained from [7].

weight pruning, and some tuning parameters are needed to remove the redundant weights. Third, freezing of the dependent nodes at each iteration has no effect on the networks' output. Therefore, the training error function will decrease monotonically during the training process in our algorithm. In [37], the convergence of the algorithm depends on some tuning parameters. The training error function may be increased so much that the training process may diverge.

In [17], the firing similarity r_{ij} of two hidden units is evaluated,

$$r_{ij} = \sum_{p=1}^{P}(o_{ip}-\bar{o}_i)(o_{jp}-\bar{o}_j) \Bigg/ \left[\sum_{p=1}^{P}(o_{ip}-\bar{o}_i)^2\sum_{p=1}^{P}(o_{jp}-\bar{o}_j)^2\right]^{1/2}.$$

Here o and $\bar{o}$ denote the output and the mean output of a unit, respectively. Similarity r_{ij} falls between -1 and 1. Synchronous pairs ($r_{ij}>0$) and contrary pairs ($r_{ij}<0$) of hidden units are pruned by fusing method. The neurons are pruned in a decreasing order of $\|r_{ij}\|$. Since the fused neurons with $\|r_{ij}\|\neq 1$ will increase the error, retraining is needed. Our approach differs from the fusing method in [17] in the following aspects. First, in this proposed algorithm, pruning occurs during training, rather than after training. Second, in this algorithm, only units with $\|r_{ij}\|=1$ will be fused, the error is maintained and no retraining is needed. Third, the similarity between two nodes is extended. As long as the output of a node is in the space spanned by the other nodes, it will be pruned. Finally, nodes with constant output will be fused with the bias unit, this is not covered in [17]. In [29,30], three categories of excessive hidden nodes, called constant-output nodes, parallel nodes and anti-parallel nodes, are defined, and a heuristic specific rule is devised for locating redundant nodes. All these three categories of redundant nodes are covered by dependent nodes of cases 1 and 2 in the analysis in this section. As discussed in [2], the approach in [29,30] involves determining a number of problem dependent threshold parameters to select the dependent nodes. However, those choices on thresholds are problematic: small threshold values, in fact, typically lead to removing very few redundant units, while large values result in too many excised nodes, thereby seriously worsening the performance of the network. Due to these approximations, a further slow retraining stage is generally required after pruning and sometimes the retraining process may even fail to converge [2]. The redundant nodes identification approach used in our algorithm is similar to the approach in [29,30]. However, rather than locating and pruning the redundant nodes after training, the dependent nodes are kept frozen at each iteration during the training process to maintain the dependence as much as possible. Therefore, after training, more nodes are kept dependent and can be pruned using a very small threshold ρ (in our simulation, $\rho = 1.0\mathrm{e}-4$). As a result, the performance of the network is maintained, and no retraining is needed. This will be demonstrated in the simulation.

4. Unit-based optimal brain surgeon pruning

In this section, a UB-OBS pruning method is described, which prunes hidden units in feedforward NNs.

Consider a network trained by the training and pruning algorithm in the last section to a local minimum of the error function V in Eq. (5).

First, the dependent hidden nodes are identified using QR factorization, and then prune them and recalculate the output weights of the remaining nodes to maintain the input–output behavior of the network. Pruning of the dependent nodes will not change the error function V.

For notation simplicity, in the rest of this section, the pruned network is described as

$$\Phi W^{\mathrm{T}} = Y, \tag{37}$$

where Φ is a $N\times r$ full column rank matrix, and the remaining weight W can be easily obtained as

$$W = Y^{\mathrm{T}}\Phi(\Phi^{\mathrm{T}}\Phi)^{-1}. \tag{38}$$

Next, a novel sensitivity measure is proposed to remove the insensitive nodes based on the procedure developed by Hassibi and Stock [13].

Pruning the jth hidden node is equivalent to setting the corresponding weights from the jth node to all the output nodes to be zeros, that is $W_{ij}=0$, $i=1,\dots,p$. The new weight W after pruning needs to be updated such that the output of ΦW^{T} remains as close as possible to Y.

$$\begin{aligned} V &= \mathrm{tr}\{(Y-\Phi W^{\mathrm{T}})^{\mathrm{T}}(Y-\Phi W^{\mathrm{T}})\} \\ &= \mathrm{tr}\{Y^{\mathrm{T}}Y - 2Y^{\mathrm{T}}\Phi W^{\mathrm{T}} + W\Phi^{\mathrm{T}}\Phi W^{\mathrm{T}}\}. \end{aligned} \tag{39}$$

The functional Taylor series of the error with respect to the output weights is

$$\delta V = \left(\frac{\partial V}{\partial W}\right)\delta W + \frac{1}{2}\delta W^{\mathrm{T}}\cdot H\cdot\delta W + O(\|\delta W\|^3), \tag{40}$$

where $H = \partial^2 V/\partial^2 W = \Phi^{\mathrm{T}}\Phi$ is the Hessian matrix. The first term vanishes for a network trained to a local minimum, and the third term is zero.

The updating process can be formulated as an optimization problem:

$$\min_{\delta W} \quad \delta V$$
$$\text{s.t.} \quad \delta W e_j = -W_j, \tag{41}$$

where e_j and W_j are the jth column of the identity matrix and W, respectively, that is $W_j = We_j$. The constraint in (41) means that the jth node is assumed to be pruned.

Solving the constrained linear programming problem in (41) using the Lagrangian method,

$$\min_{\delta W} \delta V = \mathrm{tr}\{\delta W \Phi^{\mathrm{T}} \Phi \delta W^{\mathrm{T}}\} + \lambda(\delta W e_j + W_j)^{\mathrm{T}}(\delta W e_j + W_j), \tag{42}$$

where λ is the Lagrange undetermined multiplier. After taking functional derivatives, it can be shown that

$$\delta W = -\lambda(\delta W e_j + W_j)e_j^{\mathrm{T}}(\Phi^{\mathrm{T}}\Phi)^{-1}. \tag{43}$$

To solve λ, both sides of Eq. (43) are multiplied by e_j to obtain

$$\lambda(\delta W e_j + W_j) = -\delta W e_j/(\Phi^{\mathrm{T}}\Phi)^{-1}_{jj} = W_j/(\Phi^{\mathrm{T}}\Phi)^{-1}_{jj}. \tag{44}$$

Here $(\Phi^{\mathrm{T}}\Phi)^{-1}_{jj} = e_j^{\mathrm{T}}(\Phi^{\mathrm{T}}\Phi)^{-1}e_j$ is the jth diagonal element of $(\Phi^{\mathrm{T}}\Phi)^{-1}$. Then δW and δV can be obtained as follows:

$$\delta W = -W_j e_j^{\mathrm{T}}(\Phi^{\mathrm{T}}\Phi)^{-1}/(\Phi^{\mathrm{T}}\Phi)^{-1}_{jj}, \tag{45}$$

$$\delta V = W_j^{\mathrm{T}} W_j/(\Phi^{\mathrm{T}}\Phi)^{-1}_{jj}. \tag{46}$$

Therefore, the sensitivity measure of the minimal effect after pruning the jth node and readjusting the remaining weights can be defined as

$$\mathrm{mes}_j := W_j^{\mathrm{T}} W_j/(\Phi^{\mathrm{T}}\Phi)^{-1}_{jj}. \tag{47}$$

The lower the value of mes_j, the less effect on the output. Hence an insensitive j^*th node to be pruned can be chosen according to Eq. (47) and $\hat{W}$ can be calculated using Eq. (45). Notice that Φ is a $N \times r$ full column rank matrix after first-stage pruning, hence $\Phi^{\mathrm{T}}\Phi$ is invertible. In [2], no first-stage pruning is done, and Φ is not full column rank, therefore, the conjugate gradient algorithm is required to solve the linear system.

The procedure can be summarized as follows:

(1) The number of the remaining nodes $h := r$.
(2) Calculate mes_j for $j = 1, \ldots, h$.
Select the insensitive node j^* by Eq. (47) such that the minimal effect is $\mathrm{mes}_{j^*} = \min_j\{\mathrm{mes}_j\}$.
(3) Calculate the summed square error (SSE) V.
If SSE does not satisfy the learning accuracy requirement, then goto (5). Otherwise, calculate the $\hat{W}$ using Eq. (45).
(4) Remove the j^* node, i.e., delete the j^* column of Φ and $\hat{W}$, respectively.
Define a permutation matrix
$P := [e_1, \ldots, e_{j^*-1}, e_{j^*+1}, \ldots, e_{h-1}, e_h]$.
Set $\Phi := \Phi P$, $W := \hat{W} P$, $Y = \Phi W^{\mathrm{T}}$, $h := h-1$, goto (2).
(5) No more nodes can be pruned, stop pruning.

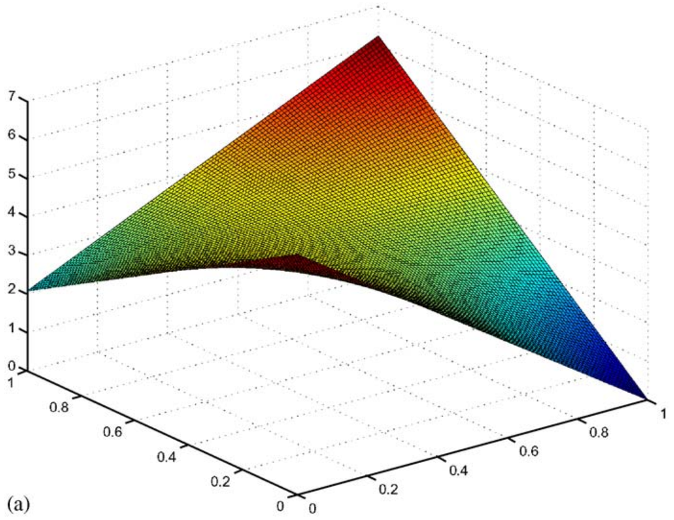
To compare with the sensitivity measure used in [2], the following analysis is made. Assume node j is to be pruned, only W_j is set to be zero and all the other columns of W are kept unchanged. Then from Eq. (37), δY can be described as

$$\delta Y = -\phi_j W_j^{\mathrm{T}}. \tag{48}$$

Therefore, the sensitivity measure of $\tilde{V}_j = \mathrm{tr}\{\delta Y^{\mathrm{T}} \delta Y\}$ is as follows:

$$\tilde{V}_j = \mathrm{tr}\{W_j \phi_j^{\mathrm{T}} \phi_j W_j^{\mathrm{T}}\} = (\phi_j^{\mathrm{T}}\phi_j)(W_j^{\mathrm{T}} W_j). \tag{49}$$

This measure $\tilde{V}_j$ is equivalent to that used in [2], which only considers the effect of pruning a hidden node without readjusting the remaining weights. The measure does not really reflect the sensitivity effect. However, the proposed measure in (47) suggests an optimal selection criterion of



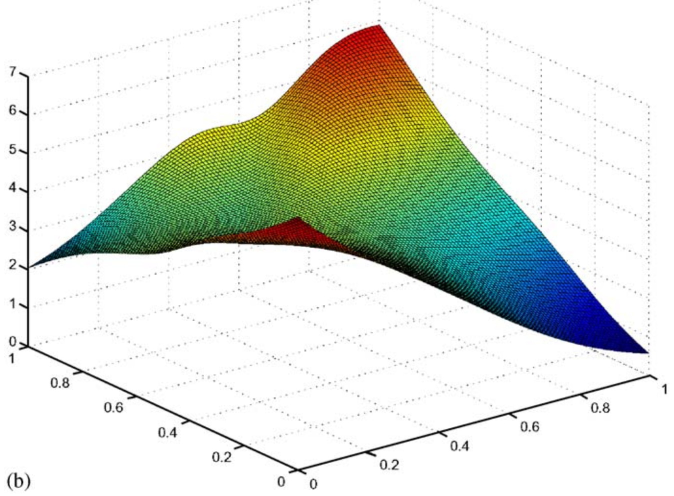


Fig. 1. Simple interaction function (SIF). (a) Original SIF. (b) Generalized SIF.

the insensitive node since it guarantees that the removal of insensitive node and readjusting the remaining weights have a minimal effect on the networks's performance.

In [3], a UB-OBS pruning algorithm is described, where eliminating a hidden unit is expressed as setting all the weights feeding to the hidden unit from the input layer to zero, therefore a different sensitivity measure and weight updating law are obtained. For a hidden layer with sigmoid activation functions, a zero input does not mean a zero output. Hence, all the output weights from the hidden unit to be eliminated are set to be zero in the algorithm proposed in this paper.

5. Simulation results

In this section, some regression problems are used to demonstrate the effectiveness of the proposed training and pruning algorithm. The performance of the network is measured by the so-called metric fraction of variance unexplained (FVU) [20], which is defined as

$$\mathrm{FVU} = \frac{\sum_{j=1}^{N} (\hat{f}(x^j) - f(x^j))^2}{\sum_{j=1}^{N} (f(x^j) - \bar{f})^2}, \tag{50}$$

where $f(.)$ is the function to be implemented by the feedforward NN, $\hat{f}(.)$ is an estimate of $f(.)$ realized by the network, and $\bar{f}$ is the mean value of $f(.)$ over the training samples. The FVU is basically the ratio of the error variance to the variance of the function being analyzed by the network. Note that the FVU is proportional to the mean square error (MSE). Furthermore, the functions under study are likely to be contaminated by an additive noise ε. In this case the signal-to-noise ratio (SNR) is defined by

$$\mathrm{SNR} = 10\log_{10}\left\{\frac{\sum_{j=1}^{N} (f(x^j) - \bar{f})^2}{\sum_{j=1}^{N} (\varepsilon^j - \bar{\varepsilon})^2}\right\}, \tag{51}$$

where $\bar{\varepsilon}$ is the mean value of the additive noise, which is usually assumed to be zero.

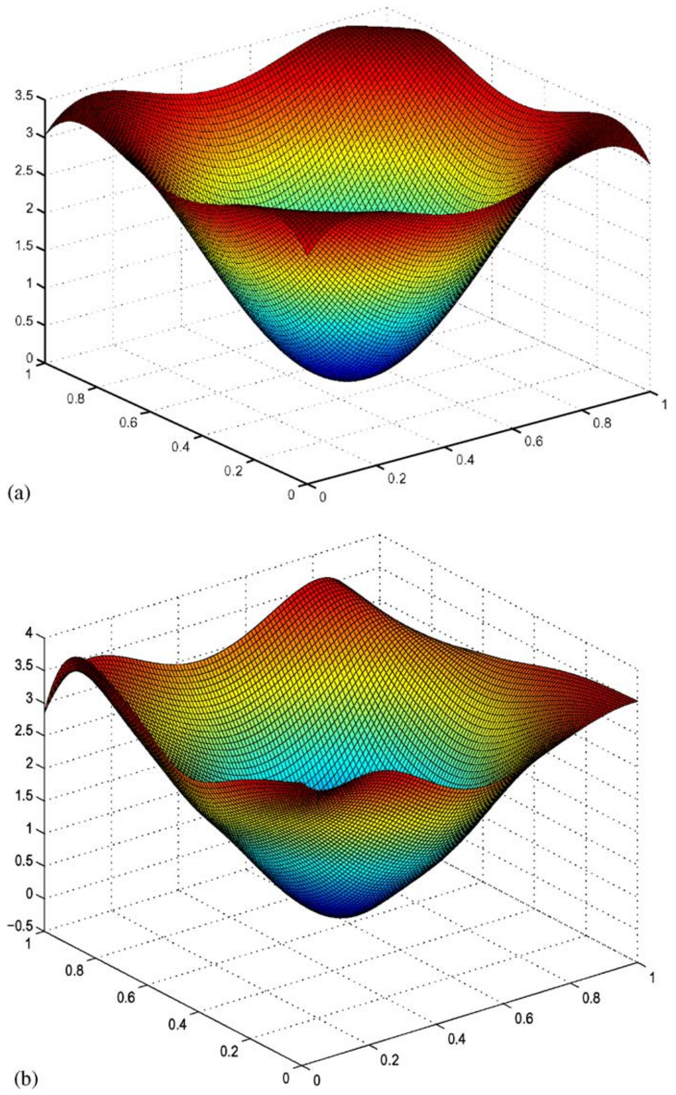


Fig. 2. Radial function (RF). (a) Original RF. (b) Generalized RF.

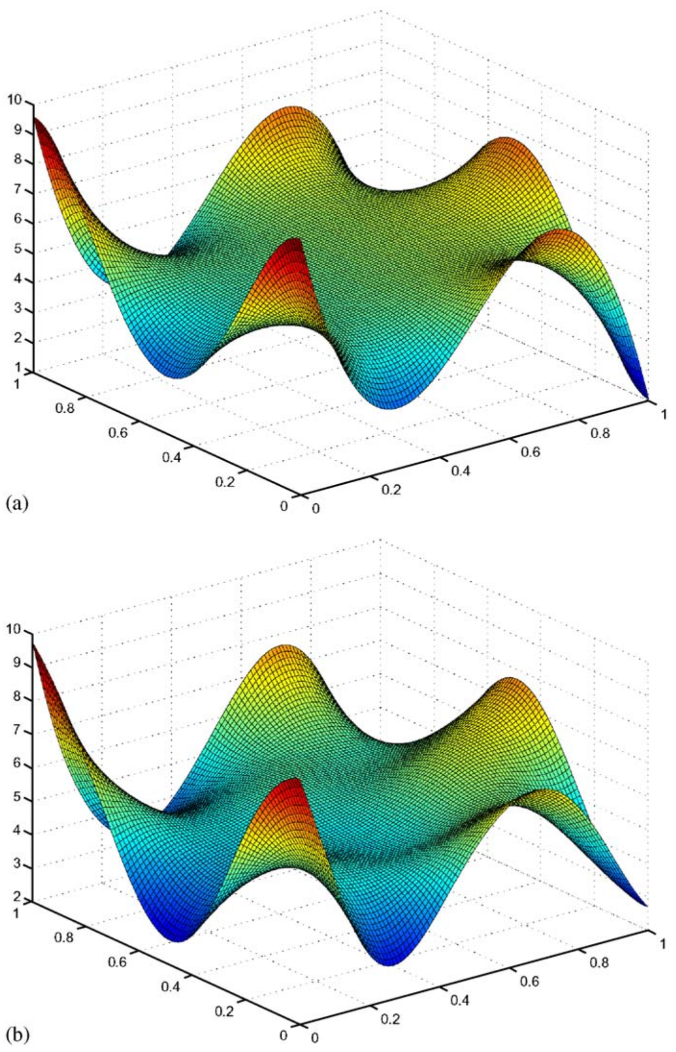


Fig. 3. Harmonic function (HF). (a) Original HF. (b) Generalized HF.

The regression problems used are the two-dimensional functions which have been used in [23,20]. They are as follows:

- Simple interaction function (SIF)

$$f^{(1)}(x_1, x_2) = 10.391((x_1 - 0.4)(x_2 - 0.6) + 0.36).$$

- Radial function (RF)

$$f^{(2)}(x_1, x_2) = 24.234(r^2(0.75 - r^2)),$$
$$r^2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2.$$

- Harmonic function (HF)

$$f^{(3)}(x_1, x_2) = 42.659((2 + x_1)/20 + Re(z^5)),$$
$$z = x_1 + ix_2 - 0.5(1 + i).$$

- Additive function (AF)

$$f^{(4)}(x_1, x_2) = 1.3356(1.5(1 - x_1) + e^{2x_1-1}\sin(3\pi(x_1 - 0.6)^2 + e^{3(x_2-0.5)}\sin(4\pi(x_2 - 0.9)^2))$$

- Complicated interaction function (CIF)

$$f^{(5)}(x_1, x_2) = 1.9(1.35 + e^{x_1}\sin(13(x_1 - 0.6)^2)e^{-x_2}\sin(7x_2)).$$

The plots of these functions are depicted in Figs. 1(a)–5(a).

For each function 225 uniformly distributed random points were generated from the two-dimensional interval [0,1] for network training. The same set of abscissa values (x's) is used for experiments with all five functions. Ten thousand uniformly sampled points from the same interval without additive noise were used to test the generalization performance of the trained network.

The training and pruning procedure of our SBTP algorithm can be summarized as follows:

Step 0: Initialize the weight parameters. The initial number of hidden nodes is $m = 41$.

Step 1: Calculate the output of each hidden neuron $\Phi(k)$. Identify the dependent nodes using QR factorization from Eq. (21). Readjust the output weights of the independent nodes by solving Eq. (28). The small positive number in

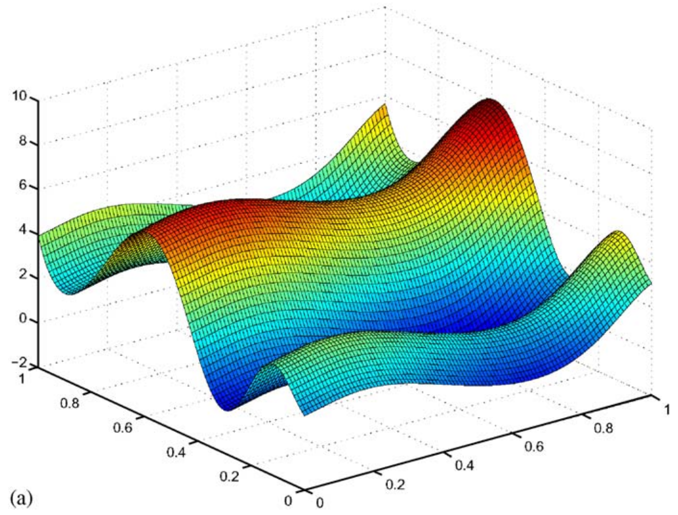
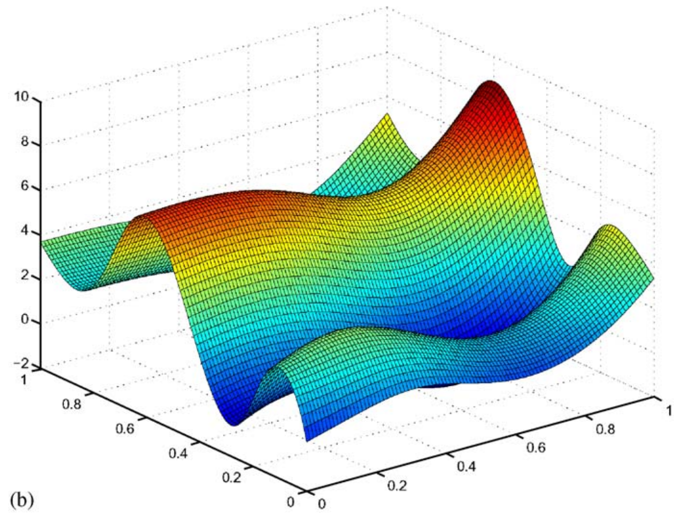



Fig. 4. Additive function (AF). (a) Original AF. (b) Generalised AF.

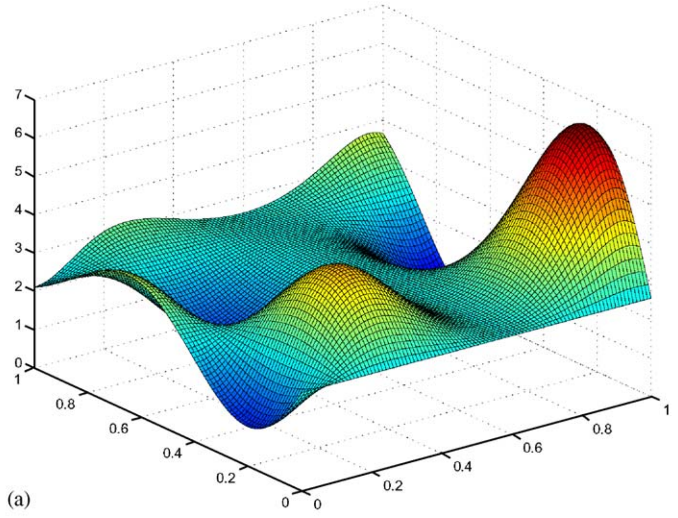
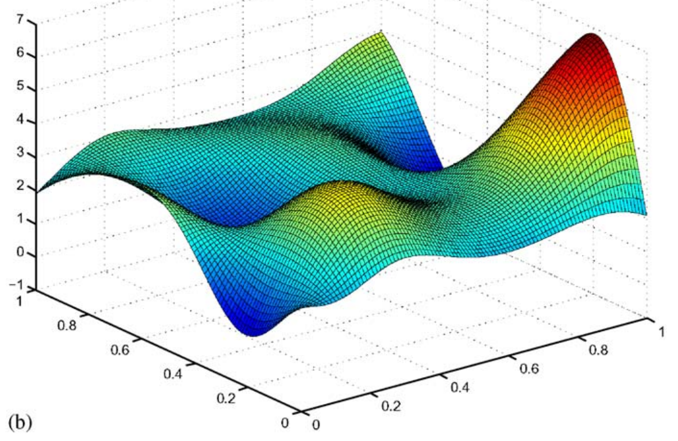



Fig. 5. Complicated interaction function (CIF). (a) Original CIF. (b) Generalised CIF.

Eq. (23) for identifying the dependent nodes is chosen as $\rho = 1.0e - 4$.

Step 2: Calculate the Jacobian matrix J^r of the independent nodes.

Step 3: Adjust the weights of the independent nodes θ^r using Eq. (36).

Step 4: Calculate $V(\theta(k+1))$. If $V(\theta(k+1)) > V(\theta(k))$, neglect the update and set $\mu := \mu\beta$, if $\mu > \mu_{\max}$, a local minimum is reached, goto step 0 to restart training using different initial weights; otherwise goto step 3. The parameters are chosen as $\mu = 1.0\mathrm{e} - 2$, $\beta = \sqrt{10}$, $\mu_{\max} = 1.0e + 10$.

Step 5: If $V(\theta(k+1)) < V(\theta(k))$, accept the update and then set $\mu := \mu/\beta$.

Step 6: If $V(\theta(k+1)) > \varepsilon$, goto step 1; Otherwise, the training objective is reached, and stop training. For a given FVU, $\varepsilon = \mathrm{FVU}\sum_{j=1}^{N}(f(x^j) - \bar{f})^2$.

Step 7: Prune the dependent nodes and insensitive nodes using the UB-OBS pruning algorithm.

Forty independent runs were performed (where for each run the network is initialized to a different set of weight values) for an ensemble-averaged evaluation for two cases where SNRs are 10 and 0 dB, respectively. Means of FVUs for the training and the generalization performance of networks for the above regression functions are summarized in Tables 2 and 3, respectively. Selected generalized surfaces of the above functions by the networks constructed using samples with 10 dB SNR are also provided for comparison in Figs. 1(b)–5(b).

In order to demonstrate the effectiveness of our proposed SBTP algorithm, the whole network is also trained using the LM algorithm with the same initial weights, and after training, the dependent and insensitive nodes are identified and pruned using the UB-OBS pruning algorithm. This procedure is called the full-set based training and pruning (FBTP) algorithm.

First, the size of the network after pruning is compared. From Tables 2 and 3, it is noticed that the number of hidden nodes of the proposed SBTP algorithm is less than that of FBTP in almost all cases for the five functions. For SIF, the number of hidden nodes of the SBTP algorithms is similar to that of FBTP. Selected plots of the number of hidden nodes at each iteration during the subset training process are shown in Fig. 6. Notice that the initial number of hidden nodes is $m = 41$. As expected, the number of independent nodes is small at the early training stage, (normally $r = 8$ or 9), which increases (to around $r = 20$) as the training proceeds to the final stage. This can be explained by the premature phenomenon.

Table 2
Simulation results (SNR = 10 dB)

Function	Approach	Number of hidden nodes after training	Number of hidden nodes after pruning	Training FVU	Generalization FVU
SIF	FBTP	41	14.97	0.0797	0.0214
	SBTP	16.89	11.93	0.0798	0.0195
RF	FBTP	41	16.70	0.0717	0.0335
	SBTP	17.71	12.83	0.0717	0.0312
HF	FBTP	41	16.93	0.0793	0.0697
	SBTP	18.43	13.53	0.0795	0.0671
AF	FBTP	41	12.43	0.0877	0.0459
	SBTP	17.56	9.73	0.0865	0.0418
CIF	FBTP	41	17.37	0.0795	0.0307
	SBTP	18.90	14.27	0.0795	0.0303

Table 3
Simulation results (SNR = 0 dB)

Function	Approach	Number of hidden nodes after training	Number of hidden nodes after pruning	Training FVU	Generalization FVU
SIF	FBTP	41	5.05	0.4760	0.1267
	SBTP	15.88	5.13	0.4738	0.1307
RF	FBTP	41	8.80	0.4949	0.1428
	SBTP	15.92	7.35	0.4946	0.1110
HF	FBTP	41	12.10	0.4930	0.3767
	SBTP	17.21	10.48	0.4943	0.4059
AF	FBTP	41	13.95	0.4456	0.2711
	SBTP	18.83	10.38	0.4466	0.2592
CIF	FBTP	41	11.25	0.4872	0.2451
	SBTP	16.22	9.28	0.4896	0.2603

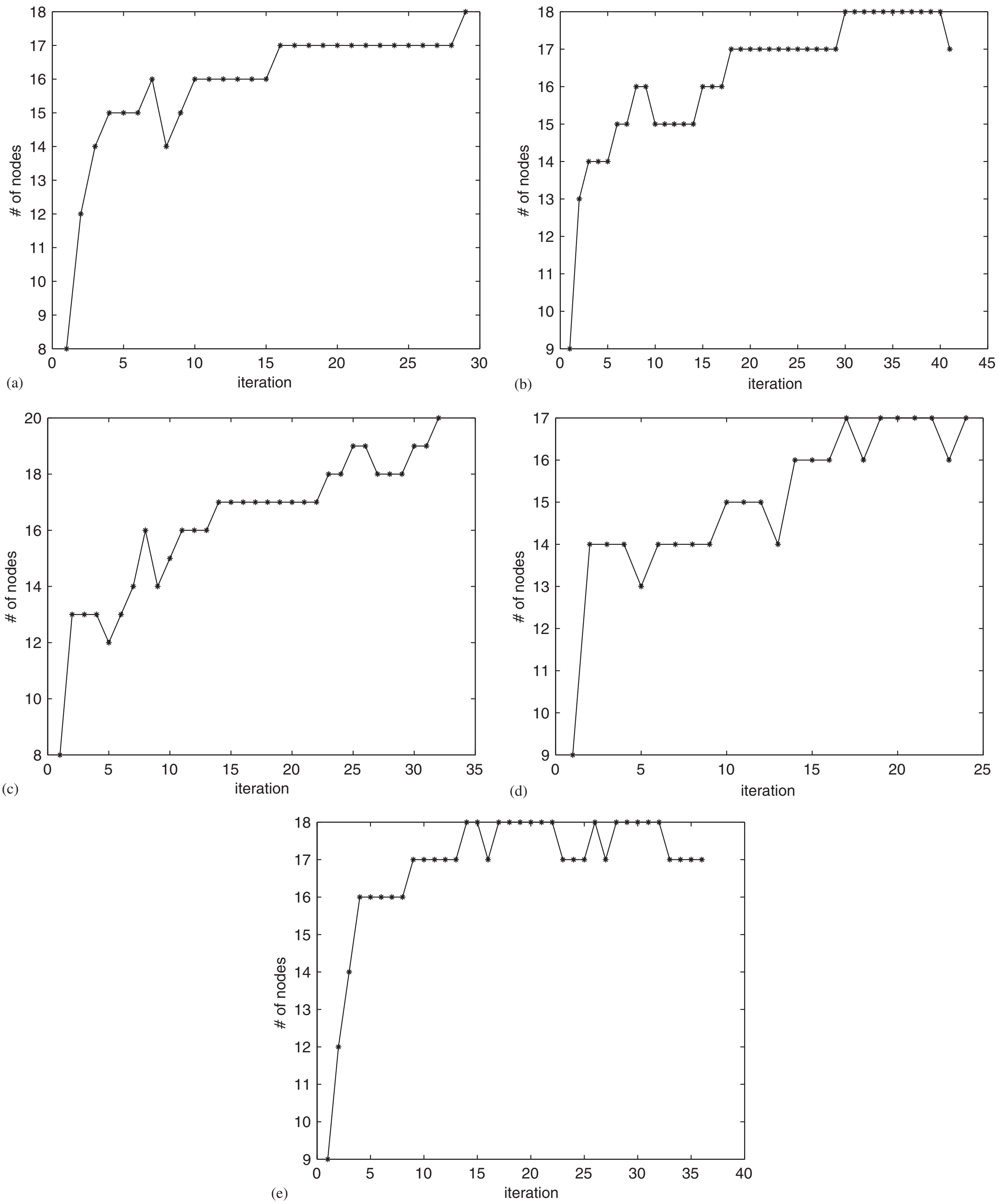


Fig. 6. The number of independent hidden nodes during the training process. (a) SIF (b) RF (c) HF (d) AF and (e) CIF.

Table 4
Comparison of the computational cost between full-set and subset training

Approach	SNR (dB)	SIF	RF	HF	AF	CIF
FBTP	10 or 0	5.4e + 6	5.4e + 6	5.4e + 6	5.4e + 6	5.4e + 6
SBTP	10	8.7e + 5	9.7e + 5	9.7e + 5	9.7e + 5	1.1e + 6
	0	7.9e + 5	7.9e + 5	8.7e + 5	1.1e + 6	7.9e + 5

Table 5
Average number of training iterations

SNR (dB)	Approach	SIF	RF	HF	AF	CIF
10	FBTP	20.14	24.76	36.88	22.50	43.26
	SBTP	31.70	44.73	47.73	23.96	62.41
0	FBTP	7.60	10.13	18.15	21.0	18.03
	SBTP	7.55	9.42	23.01	26.02	19.80

Second, the computational cost at each training iteration is compared. The number of multiplication operations of FBTP and SBTP is shown in Table 4. For all five functions, the number of hidden nodes is chosen as $m = 41$. Therefore, the computational costs are the same for the five functions using FBTP. For the proposed SBTP algorithm, the number of independent nodes is different at each iteration, as shown in Fig. 6. The number used for calculating the computational cost in Table 4 is chosen as the average number of final independent nodes after training, which is also shown in Tables 2 and 3. From Fig. 6, it can be seen that the actual number of independent nodes at each iteration is much less than the final number of independent nodes. Therefore, the actual computational cost of our algorithm is lower than that in Table 4. Nevertheless, as shown in Table 4, the computational cost of FBTP is at least five times more than that of SBTP.

Third, the generalization performance of the network is compared in Tables 2 and 3. For all functions with 10 dB SNR, the generalization FVU of the proposed SBTP algorithm is better. For RF and AF with 0 dB SNR, the generalization FVU of the proposed SBTP algorithm is better; For SIF, HF and CIF with 0 dB SNR, the generalization performance of the FBTP is better.

Finally, the convergence rate is compared. As shown in Table 5, FBTP converges to a solution in less iterations. Since there are more free weight parameters in FBTP, the learning speed is faster [2]. Selected plots of the MSE at each iteration during the training process are shown in Fig. 7.

The NNs constructed using the proposed SBTP algorithm are also compared with two previous constructive algorithms in [23,20]. It is noticed that the two previous constructive algorithms stopped training at 20 hidden nodes to achieve the training FVU. For comparison purposes, in this proposed algorithm, it is aimed to achieve the same training FVUs as those for the two previous constructive algorithms, and compare the number of hidden nodes and the generalization performances of the constructed NNs. Therefore, the training process is stopped when the training FVU is approximately the same as that for the constructive algorithms. It is noticed that the stopping criteria have some strong effect on the generalization performance. If the same stopping criteria was used as the two previous constructive algorithms, that is, the training stopped when the number of hidden nodes is 20, the training FVU would be smaller than that of the constructive algorithms, but the generalization performance would be poorer, because overfitting occurs.

Comparisons between the constructive algorithms in [23,20] and the proposed SBTP algorithm for all the five functions at 10 dB SNR are shown in Table 6. For all five functions, the number of the hidden units of the networks constructed using our proposed algorithm is much less than that of the constructive algorithms. For HF, poor results were presented in [23,20]. In fact, it is expected to achieve a better training FVU. Therefore, a lower training FVU, of 0.08 is set. It can be seen that the generalization FVU of our proposed SBTP algorithm is greatly improved to 0.0671. For SIF and CIF, the generalization performance of our SBTP algorithm is better than that for constructive algorithms. For RF and AF, our result is in between. It is can be seen that the proposed technique can prevent overfitting and provides similar generalization performance with less hidden nodes.

6. Conclusion

A training and pruning algorithm based on the concept of Jacobian deficiency is developed in this paper. The goal is to aim at reducing the training computational cost while reducing the number of networks nodes to overcome overfitting. This overcomes the drawbacks of the existing post-training pruning algorithms by reducing the heavy training cost and avoids retraining. The algorithm is quite general and can be applied to any feedforward NN. Notice

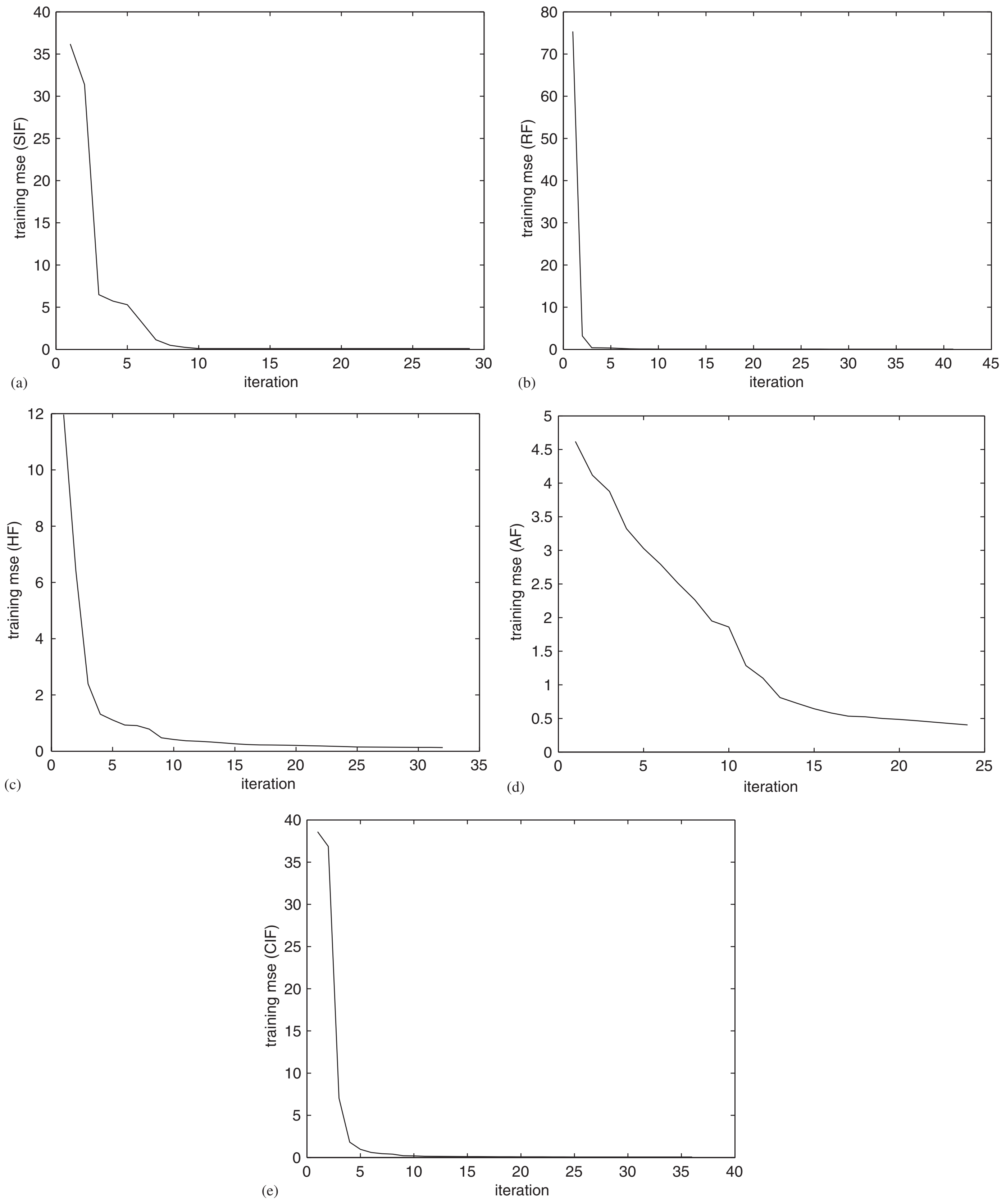


Fig. 7. The mean square error (mse) during the training process. (a) SIF (b) RF (c) HF (d) AF and (e) CIF.

Table 6
Comparison with constructive algorithms (SNR = 10 dB)

Function	Approach	Number of hidden nodes	Training FVU	Generalization FVU
SIF	[20]	20	0.082	0.032
	[23]	20	0.085	0.022
	SBTP	11.93	0.0798	0.0195
RF	[20]	20	0.072	0.036
	[23]	20	0.073	0.024
	SBTP	12.83	0.0717	0.0312
HF	[20]	20	0.171	0.290
	[23]	20	0.157	0.219
	SBTP	13.53	0.0795	0.0671
AF	[20]	20	0.092	0.048
	[23]	20	0.095	0.040
	SBTP	9.73	0.0865	0.0418
CIF	[20]	20	0.089	0.108
	[23]	20	0.095	0.088
	SBTP	14.27	0.0795	0.0303

that the redundant nodes are identified based on the linear dependency among hidden node outputs, which is mostly caused by saturation characteristics inherent in the sigmoid function. Therefore, the proposed algorithm is more suitable to MLP. For feedforward NNs with other activation functions, an efficient redundant node selection algorithm is under investigation.

References

[1] T. Ash, Dynamic node creation in backpropagation networks, Connection Sci. 4 (1989) 365–375.

[2] G. Castellano, A.M. Fanelli, M. Pelillo, An iterative pruning algorithm for feedforward neural networks, IEEE Trans. Neural Networks 3 (1997) 519–531.

[3] H. Chandrasekaran, H. Chen, M.T. Manry, Pruning of basis functions in nonlinear approximators, Neurocomputing 34 (2000) 29–53.

[4] S. Chang, K. Wong, C. Leung, Periodic activation function for fast on-line EKF training and pruning, Electron. Lett. 23 (1998) 2255–2256.

[5] F.L. Chung, T. Lee, A node pruning algorithm for backpropagation networks, Int. J. Neural Syst. 3 (1992) 301–314.

[6] S. Cohen, N. Intrator, Forward and backward selection in regression hybrid network, in: F. Roli, J. Kittler (Eds.), Lecture Notes in Computer Science, vol. 2364, Springer, Berlin, 2002, pp. 98–107.

[7] J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart, LINPACK Users' Guide, 1979.

[8] A.P. Engelbrecht, A new pruning heuristic based on variance analysis of sensitivity information, IEEE Trans. Neural Networks 12 (2001) 1386–1399.

[9] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing, vol. 2, Morgan Kaufmann, CA, 1990, pp. 524–532.

[10] L. Fletcher, V. Katkovnik, F.E. Steffens, A.P. Engelbrecht, Optimizing the number of hidden nodes of a feedforward artificial neural network, in: IEEE World Congress on Computational Intelligence, Anchorage, Alaska, 1998, pp. 1608–1612.

[11] F. Girosi, M. Jones, T. Poggio, Regularization theory and neural network architectures, Neural Comput. 7 (1995) 219–269.

[12] M.T. Hagan, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Networks 6 (1994) 989–993.

[13] B. Hassibi, D.G. Stork, Second order derivatives for network pruning: optimal brain surgeon, in: S.J. Hanson (Ed.), Advances in Neural Information Processing, vol. 5, Morgan Kaufmann, CA, 1993, pp. 164–171.

[14] J.N. Hwang, S.R. Lay, M. Maechler, D. Martin, J. Schimert, Regression modeling in backpropagation and project pursuit learning, IEEE Trans. Neural Networks 3 (1994) 342–353.

[15] M. Ishikawa, Structural learning with forgetting, Neural Networks 9 (1996) 509–521.

[16] Md.M. Islam, K. Murase, A new algorithm to design compact two-hidden-layer artificial neural networks, Neural Networks 14 (2001) 1265–1278.

[17] K. Kameyama, Y. Kosugi, Neural network pruning by fusing hidden layer units, IEICE Trans. Commun. Electron. Inf. Syst. 74 (1991) 4198–4204.

[18] P.P. Kanjilal, P.K. Dey, D.N. Banerjee, Reduced-size neural networks through singular value decomposition and subset selection, Electron. Lett. 17 (1993) 1515–1518.

[19] T. Kwok, D. Yeung, Constructive algorithms for structure learning in feedforward neural networks for regression problems, IEEE Trans. Neural Networks 3 (1997) 630–645.

[20] T. Kwok, D. Yeung, Objective functions for training new hidden units in constructive neural networks, IEEE Trans. Neural Networks 8 (1997) 1131–1148.

[21] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing, vol. 2, Morgan Kaufmann, CA, 1990, pp. 598–605.

[22] C.S. Leung, K. Wong, P.F. Sum, L.W. Chan, On-line training and pruning for recursive least square algorithm, Electron. Lett. 23 (1996) 2152–2153.

[23] L. Ma, K. Khorasani, New training strategies for constructive neural networks with application to regression problems, Neural Networks 17 (2004) 589–609.

[24] J. Platt, A resource-allocating network for function interpolation, Neural Comput. 3 (1991) 213–225.

[25] R. Reed, Pruning algorithm—a survey, IEEE Trans. Neural Networks 5 (1993) 740–747.

[26] D.W. Ruck, S.K. Rogers, M. Kabrisky, Feature selection using a multilayer perceptron, Neural Network Comput. 2 (1990) 40–48.

[27] C. Schittenkopf, G. Deco, W. Brauer, Two strategies to avoid overfitting in feedforward neural networks, Neural Networks 10 (1997) 505–516.

[28] R. Setiono, A penalty-function approach for pruning feedforward neural networks, Neural Comput. 9 (1997) 185–204.

[29] J. Sietsma, R.J.F. Dow, Neural net pruning: why and how, in: Proceedings of the IEEE International Conference on Neural Networks, vol. 1, San Diego, CA, 1988, pp. 325–333.

[30] J. Sietsma, R.J.F. Dow, Creating artificial neural networks that generalize, Neural Networks 4 (1991) 67–79.

[31] K. Suzuki, Determining the receptive field of a neural filter, J. Neural Eng. 1 (2004) 228–237.

[32] K. Suzuki, I. Horiba, N. Sugie, A simple neural network pruning algorithm with application to filter synthesis, Neural Process. Lett. 13 (2001) 43–53.

[33] M.F. Tenorio, W.T. Lee, Self-organizing network for optimum supervised learning, IEEE Trans. Neural Networks 1 (1990) 100–110.

[34] J.E. Vitela, J. Reifman, Premature saturation in back propagation networks: mechanism and necessary condition, Neural Networks 10 (1997) 721–735.

[35] Y. Wang, C. Lin, A second-order learning algorithm for multilayer networks based on block Hessian matrix, Neural Networks 11 (1998) 1607–1622.

[36] G. Zhou, J. Si, Advanced neural network training algorithm with reduced complexity based on Jacobian deficiency, IEEE Trans. Neural Networks 3 (1998) 448–453.

[37] G. Zhou, J. Si, Subset-based training and pruning of sigmoid neural networks, Neural Networks 12 (1999) 79–89.
[38] J.M. Zurada, A. Malinowski, S. Usui, Perturbation method for deleting redundant inputs of perceptron networks, Neurocomputing 14 (1997) 177–193.

Jinhua Xu received the B.S. and M.S. degrees from Xi'an Jiaotong University, Xi'an, China, in 1988 and 1991, respectively, and the Ph.D. degree from City University of Hong Kong, in 2000. She joined the department of Computer Science, East China Normal University, Shanghai, China in 2003. Her current research interests include neural networks, signal processing, genetic algorithms.

Dr. Daniel W. C. Ho received first class B.Sc., M.Sc. and Ph.D. degrees in mathematics from the University of Salford (UK) in 1980, 1982 and 1986, respectively. From 1985 to 1988, Dr. Ho was a Research Fellow in Industrial Control Unit, University of Strathclyde (Glasgow, Scotland). In 1989, he joined the Department of Mathematics, City University of Hong Kong, where he is currently an Associate Professor. He is now serving as an Associate Editor for Asian Journal of Control. His research interests include H-infinity control theory, robust pole assignment problem, adaptive neural wavelet identification, nonlinear control theory.