Neural Comput & Applic (2012) 21:1709–1716
DOI 10.1007/s00521-012-0832-6

ORIGINAL ARTICLE

Decouple implementation of weight decay for recursive least square

Andrew Chi-Sing Leung · Yi Xiao · Yong Xu · Kwok-Wo Wong

Received: 6 September 2010 / Accepted: 5 January 2012 / Published online: 1 February 2012

Abstract In the conventional recursive least square (RLS) algorithm for multilayer feedforward neural networks, controlling the initial error covariance matrix can limit weight magnitude. However, the weight decay effect decreases linearly as the number of learning epochs increases. Although we can modify the original RLS algorithm to maintain a constant weight decay effect, the computational and space complexities of the modified RLS algorithm are very high. This paper first presents a set of more compact RLS equations for this modified RLS algorithm. Afterwards, to reduce the computational and space complexities, we propose a decoupled version for this algorithm. The effectiveness of this decoupled algorithm is demonstrated by computer simulations.

Keywords Weight decay · Regularization · Recursive least square

1 Introduction

In the past two decades, the recursive least square (RLS) algorithm and the extended Kalman filter (EKF) algorithm in training multilayered feedforward neural networks (MFNNs) have been extensively investigated [1–5]. RLS or EKF algorithms belong to the *online* mode approach in which the weights are updated immediately after the presentation of a training pattern. The advantages of the online mode training approach are that it does not require the storage of the entire input output history and that in conjunction with the use of a weight decay factor, it can be used to estimate processes which are "mildly" non-stationary. RLS and EKF algorithms are efficient second-order gradient descent training methods. Compared to first-order methods, such as the backpropagation (BP) algorithm [6], they have a faster convergence rate. Moreover, in the RLS and EKF algorithms, fewer parameters are required to be tuned during the training.

Leung et al. [1, 2] found that the RLS algorithm has an implicit weight decay term [7–9] by controlling the initial value of the error covariance matrix. With the weight decay term, the magnitudes of the trained weights are constrained to be small. Hence, the network output function is smooth and the generalization ability is improved. Besides, when magnitudes of the trained weights are small, the effect of weight faults can be suppressed [10, 11].

However, the weight decay effect in the standard RLS is not substantial and decreases when the number of training cycles increases. That means, the generalization ability of the network trained with this algorithm is not fully controllable. By tackling this problem, a constant true weight decay RLS algorithm, namely true weight decay RLS (TWDRLS), was proposed [12]. The TWDRLS algorithm is able to make the weights decay effect more effective. Consequently, a network trained with this algorithm exhibits a better generalization ability. However, the computational complexity of the TWDRLS algorithm is equal to $O(M^3)$ which is much higher than that of the standard RLS, i.e., $O(M^2)$, where M is the number of weights in the network. Therefore, it is necessary to reduce the complexity of this algorithm so that the TWDRLS algorithm can be applied to large scale practical problems.

This paper first derives a set of concise equations for the TWDRLS algorithm and discusses the decay effect of the algorithm in this form. The main contribution of this paper

A. C.-S. Leung (✉) · Y. Xiao · Y. Xu · K.-W. Wong
Department of Electronic Engineering,
City University of Hong Kong, Kowloon, Hong Kong
e-mail: eeleungc@cityu.edu.hk

is to propose a decoupled version for the TWDRLS algorithm. The goal is to reduce both the computational complexity and storage requirement. In this decoupled version, instead of using one set of TWDRLS equations to train all weights, each neuron, except input neurons, is associated with a set of decoupled TWDRLS equations, which is used for training its corresponding input weights only. The overall complexity of all decoupled sets of TWDRLS equations is very low.

The rest of the paper is organized as follows. In Sect. 2, we give a brief review on the RLS and TWDRLS algorithms. We describe the decoupled TWDRLS algorithm in Sect. 3. The computer simulation results are presented in Sect. 4. Finally, we summarize our findings in Sect. 5.

2 TWDRLS algorithm

Since a MFNN with a single hidden layer that has sufficient hidden neurons is able to approximate any function [13], we consider that the MFNN model has n_o output nodes, n_h hidden nodes, and n_{in} input nodes. The output of the ith neuron in the output layer is denoted by y_i^o. The output of the jth element of the hidden layer is denoted by y_j^h. The kth element of the input $\mathbf{x}$ of the network is denoted by y_i^{in}.

The connection weight from the jth hidden neuron to the ith output neuron is denoted by $w_{i,j}^o$. Output biases are implemented as weights and are denoted by w_{i,n_h+1}^o. The connection weight from the kth input to the jth hidden neuron is denoted by $w_{j,k}^{\text{in}}$. Input biases are implemented as weights and are denoted by $w_{j,n_{\text{in}}+1}^o$. The total number of weights in the network is equal to

$$M = n_o(n_h + 1) + n_h(n_{\text{in}} + 1). \tag{1}$$

Using the conventional notation, in the RLS approach [14, 15], we arrange all weights in a M-dimensional vector, given by

$$\boldsymbol{w} = \left[w_{1,1}^{\text{in}}, \ldots, w_{n_h,(n_{\text{in}}+1)}^{\text{in}}, w_{1,1}^o, \ldots, w_{n_o,(n_h+1)}^o\right]^T. \tag{2}$$

In the RLS approach, the objective function at the tth training iteration is to minimize the following energy function:

$$E(\boldsymbol{w}) = \sum_{\tau=1}^{t} [\boldsymbol{d}(\tau) - \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(\tau))]^T [\boldsymbol{d}(\tau) - \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(\tau))] + [\boldsymbol{w} - \hat{\boldsymbol{w}}(0)]^T \boldsymbol{P}^{-1}(0)[\boldsymbol{w} - \hat{\boldsymbol{w}}(0)] \tag{3}$$

where $\{\boldsymbol{x}(\tau), \boldsymbol{d}(\tau)\}$ is the training input-out pair at the τth training iteration, $\boldsymbol{h}(\cdot,\cdot)$ is an n_o-dimensional nonlinear function describing the network. The matrix $\boldsymbol{P}(0)$ is the initial error covariance matrix in the RLS algorithm and it is usually set to δ^{-1} $\boldsymbol{I}_{M\times M}$, where $\boldsymbol{I}_{M\times M}$ is an $M \times M$ identity matrix. The magnitude of the initial weight vector $\hat{\boldsymbol{w}}(0)$ should be small.

The minimization of (3) results in the following recursive equations [14, 15]:

$$\boldsymbol{K}(t) = \boldsymbol{P}(t-1)\boldsymbol{H}^T(t)\left[I_{n_L\times n_L} + \boldsymbol{H}(t)\boldsymbol{P}(t-1)\boldsymbol{H}^T(t)\right]^{-1} \tag{4}$$

$$\boldsymbol{P}(t) = \boldsymbol{P}(t-1) - \boldsymbol{K}(t)\boldsymbol{H}(t)\boldsymbol{P}(t-1) \tag{5}$$

$$\hat{\boldsymbol{w}}(t) = \hat{\boldsymbol{w}}(t-1) + \boldsymbol{K}(t)[\boldsymbol{d}(t) - \boldsymbol{h}(\hat{\boldsymbol{w}}(t-1), \boldsymbol{x}(t))], \tag{6}$$

where

$$\boldsymbol{H}(t) = \left[\left.\frac{\partial \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(t))}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\hat{\boldsymbol{w}}(t-1)}\right]^T, \tag{7}$$

is the gradient matrix of $\boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(t))$ with size $n_o{\times}M$; and $\boldsymbol{K}(t)$ is the so-called Kalman gain matrix (with size $M \times n_o$). The matrix $\boldsymbol{P}(t)$ is the so-called error covariance matrix and is symmetric positive definite.

From (3), the standard RLS algorithm has a weight decay term $\boldsymbol{w}^T\boldsymbol{P}^{-1}(0)\boldsymbol{w}$. However, as mentioned in [1, 2], the standard RLS algorithm only has the limited weight decaying effect which is equal to $\frac{\delta}{t_o}$ per training iteration, where t_o is the number of training iterations. That is, the effect of the weight decay term in each training iteration decreases linearly as the number of training iterations increases. Hence, the more training presentations take place, the less smoothing effect would have in the data fitting process. When the number of presentations cannot be determined a priori, using the value of δ to control the generalization ability becomes impractical.

In [12], the TWDRLS algorithm was proposed to enhance the weight decay effect. In this algorithm, a new energy function was considered, given by

$$E(\boldsymbol{w}) = \sum_{\tau=1}^{t}\left[[\boldsymbol{d}(\tau) - \boldsymbol{h}(\boldsymbol{w},\boldsymbol{x}(\tau))]^T[\boldsymbol{d}(\tau) - \boldsymbol{h}(\boldsymbol{w},\boldsymbol{x}(\tau))] + \alpha\boldsymbol{w}^T\boldsymbol{w}\right] + [\boldsymbol{w} - \hat{\boldsymbol{w}}(0)]^T\tilde{\boldsymbol{P}}^{-1}(0)[\boldsymbol{w} - \hat{\boldsymbol{w}}(0)] \tag{8}$$

where α is a regularization parameter, the matrix $\tilde{\boldsymbol{P}}(0)$ is the initial error covariance matrix is usually set to δ^{-1} $\boldsymbol{I}_{M\times M}$. In (8), there is a constant decay term $\alpha\boldsymbol{w}^T\boldsymbol{w}$, and the decay effect per each training iteration does not decrease as the number of training iterations increases.

The gradient of this energy function in (8) is given by

$$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} \approx 2\tilde{\boldsymbol{P}}^{-1}(0)[\boldsymbol{w} - \hat{\boldsymbol{w}}(0)] + 2\sum_{\tau=1}^{t}\left[\alpha\boldsymbol{w} - \boldsymbol{H}^T(\tau)[\boldsymbol{d}(\tau) - \boldsymbol{H}(\tau)\boldsymbol{w} - \xi(\tau)]\right]. \tag{9}$$

In the above, we use the common linearization technique in RLS [4, 5]. That is, we linearize $\boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}\ (\tau))$ around the estimate $\hat{\boldsymbol{w}}(\tau - 1)$, given by

$$\boldsymbol{h}(\boldsymbol{w}, \vec{x}(\tau)) = \boldsymbol{H}(\tau)\boldsymbol{w} + \xi(\tau) \tag{10}$$

where

$$\boldsymbol{H}(\tau) = \left[\frac{\partial \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(\tau))}{\partial \boldsymbol{w}} \bigg|_{\boldsymbol{w}=\hat{\boldsymbol{w}}(\tau-1)} \right]^T, \tag{11}$$

$$\xi(\tau) = \boldsymbol{h}(\hat{\boldsymbol{w}}(\tau-1), \boldsymbol{x}(\tau)) - \boldsymbol{H}(\tau)\hat{\boldsymbol{w}}(\tau) + \rho(\tau) \tag{12}$$

is the residual in the expansion of $\boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}\ (\tau))$, and $\rho(\tau)$ consists of higher order residual. In the derivation, we assume that the higher order residual is not a function of $\boldsymbol{w}$. This assumption is commonly used in the derivation of the RLS or EKF equations [4, 5].

To minimize the energy function, we set the gradient to zero. Hence, we have

$$\hat{\boldsymbol{w}}(t) = \tilde{\boldsymbol{P}}(t)\boldsymbol{r}(t) \tag{13}$$

where

$$\tilde{\boldsymbol{P}}^{-1}(t) = \tilde{\boldsymbol{P}}^{-1}(0) + \sum_{\tau=1}^{t} \left[\boldsymbol{H}^T(\tau)\boldsymbol{H}(\tau) + \alpha \boldsymbol{I}_{M\times M}\right] \tag{14}$$

$$= \tilde{\boldsymbol{P}}^{-1}(t-1) + \boldsymbol{H}^T(t)\boldsymbol{H}(t) + \alpha \boldsymbol{I}_{M\times M} \tag{15}$$

$$\boldsymbol{r}(t) = \tilde{\boldsymbol{P}}^{-1}(0)\hat{\boldsymbol{w}}(0) + \sum_{\tau=1}^{t} \boldsymbol{H}^T(\tau)[\boldsymbol{d}(\tau) - \xi(\tau)] \tag{16}$$

$$= \boldsymbol{r}(t-1) + \boldsymbol{H}^T(t)[\boldsymbol{d}(t) - \xi(t)]. \tag{17}$$

Furthermore, define

$$\boldsymbol{P}^*(t) \triangleq \left[\boldsymbol{I}_{M\times M} + \alpha\tilde{\boldsymbol{P}}(t-1)\right]^{-1}\tilde{\boldsymbol{P}}(t-1). \tag{18}$$

Hence, we have

$$\boldsymbol{P}^{*-1}(t) = \tilde{\boldsymbol{P}}^{-1}(t-1) + \alpha \boldsymbol{I}_{M\times M}. \tag{19}$$

Note that

$$\left[\left[\boldsymbol{I}_{M\times M} + \alpha\tilde{\boldsymbol{P}}(t-1)\right]^{-1}\tilde{\boldsymbol{P}}(t-1)\right]\left[\tilde{\boldsymbol{P}}^{-1}(t-1) + \alpha \boldsymbol{I}_{M\times M}\right] = \boldsymbol{I}_{M\times M}. \tag{20}$$

Employing the matrix inversion lemma:

$$(\boldsymbol{A}^{-1} + \boldsymbol{B}\boldsymbol{C}^{-1}\boldsymbol{B}^T)^{-1} = \boldsymbol{A} - \boldsymbol{A}\boldsymbol{B}(\boldsymbol{C} + \boldsymbol{B}^T\boldsymbol{A}\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{A}, \tag{21}$$

in the recursive calculation of $\boldsymbol{P}(t)$, (13) becomes the following recursive equations:

$$\boldsymbol{P}^*(t-1) = \left[\boldsymbol{I}_{M\times M} + \alpha\tilde{\boldsymbol{P}}(t-1)\right]^{-1}\tilde{\boldsymbol{P}}(t-1) \tag{22}$$

$$\boldsymbol{K}(t) = \boldsymbol{P}^*(t-1)\boldsymbol{H}^T(t)\left[\boldsymbol{I}_{n_o\times n_o} + \boldsymbol{H}(t)\boldsymbol{P}^*(t-1)\boldsymbol{H}^T(t)\right]^{-1} \tag{23}$$

$$\tilde{\boldsymbol{P}}(t) = \boldsymbol{P}^*(t-1) - \boldsymbol{K}(t)\boldsymbol{H}(t)\boldsymbol{P}^*(t-1) \tag{24}$$

$$\hat{\boldsymbol{w}}(t) = \hat{\boldsymbol{w}}(t-1) - \alpha\tilde{\boldsymbol{P}}(t)\hat{\boldsymbol{w}}(t-1) + \boldsymbol{K}(t)\left[\vec{d}(t) - \boldsymbol{h}(\hat{\boldsymbol{w}}(t-1), \boldsymbol{x}(t))\right]. \tag{25}$$

(22)–(25) are the general global true weight decay recursive equations. They are more compact than the equations presented in [12].

In (22)–(25), we can easily observe that when the regularization parameter α is set to zero, the term $\alpha\tilde{\boldsymbol{P}}(t)\hat{\boldsymbol{w}}(t-1)$ vanishes in (25), and (22)–(25) reduce to the standard RLS equations. We note that the main difference between the standard RLS equations and the TWDRLS equations is the introduction of a weight decaying term $-\alpha\tilde{\boldsymbol{P}}(t)\hat{\boldsymbol{w}}(t-1)$ in (25). The inclusion of this term guarantees that the magnitude of the updating weight vector decays an amount proportional to $\alpha\tilde{\boldsymbol{P}}(t)$. It should be notice that from the definition, $\tilde{\boldsymbol{P}}(t)$ is positive definite. Therefore, the magnitude of the weight vector would not be too large. So the generalization ability of the trained networks would be better.

We can also explain the weight decay effect from the energy function, given by (8). Clearly, when the regularization parameter α is set to zero, the energy function of the TWDRLS in (8) becomes the energy function of the standard RLS, given by (3).

From the energy function point of view, the objective of the TWDRLS is the same as that of batch model weight decay methods. Hence, existing heuristic methods [7, 16–18] for choosing the value of α can be used for the TWDRLS's case. Those methods can be applied to any training algorithms whose cost function contains a quadratic weight decay term. The most simple method is the test set validation method [7], in which we use a test set to select the most suitable value of the regularization parameter. Since the aim of this paper is to develop the RLS equations for the weight decay regularizer rather than to develop some Bayesian theories for model selection [19–22], in this paper we suggest that we train a number of networks with different values of α and then we select a network based on a test set.

A drawback of the TWDRLS algorithm is the requirement in computing the inverse of the M-dimensional matrix $(\boldsymbol{I}_{M\times M} + \alpha\tilde{\boldsymbol{P}}(t-1))$. This complexity is equal to $O(M^3)$ which is much higher than that of the standard RLS, $O(M^2)$. The TWDRLS algorithm is computationally prohibitive even for a network with moderate size. In the next Section, a decoupled version of the TWDRLS algorithm will be proposed to solve this high complexity problem.

3 Decouple the TWDRLS algorithm

3.1 Derivation

In order to decouple the TWDRLS algorithm, we first divide the weight vector into several smaller local groups.

For the *i*th output neuron, we use a decoupled weight vector

$$\boldsymbol{w}_i^o = \left[w_{i,1}^o, \ldots, w_{i,(n_h+1)}^o\right]^T \tag{26}$$

to represent those weights connecting hidden neurons to the *i*th output neuron. For the *j*th hidden neuron, we use a decoupled weight vector

$$\boldsymbol{w}_j^{\text{in}} = \left[w_{j,1}^{\text{in}}, \ldots, w_{j,(n_{\text{in}}+1)}^{\text{in}}\right]^T \tag{27}$$

to represent those weights connecting inputs to the *j*th hidden neuron.

In the decoupled version of the TWRLS algorithm, we consider the estimation of each decoupled weight vector separately. When we consider the weight vector of a neuron, we assume that other decoupled weight vectors are constant vectors[1].

For the *i*th output neuron, it is associated with a decoupled weight vector $\boldsymbol{w}_i^o$. Since we assume that other decoupled weight vectors are constant vectors, the energy function of that decoupled weight vector is given by

$$\begin{aligned} E(\boldsymbol{w}_i^o) = &\sum_{\tau=1}^{t}\left[[d_i(\tau) - h_i(\boldsymbol{w}, \boldsymbol{x}(\tau))]^2 + \alpha \boldsymbol{w}_i^{o^T}\boldsymbol{w}_i^o\right] \\ &+ \left[\boldsymbol{w}_i^o - \hat{\boldsymbol{w}}_i^o(0)\right]^T \boldsymbol{P}_i^{o^{-1}}(0)\left[\boldsymbol{w}_i^o - \hat{\boldsymbol{w}}_i^o(0)\right]. \end{aligned} \tag{28}$$

Utilizing a derivative process similar to the previous analysis, we obtain the following recursive equations for the output neurons:

$$\boldsymbol{P}_i^{o*}(t-1) = \left[\boldsymbol{I}_{(n_h+1)\times(n_h+1)} + \alpha \boldsymbol{P}_i^o(t-1)\right]^{-1}\boldsymbol{P}_i^o(t-1) \tag{29}$$

$$\boldsymbol{K}_i^o(t) = \boldsymbol{P}_i^{o*}(t-1)\boldsymbol{H}_i^{o^T}(t)\left[1 + \boldsymbol{H}_i^o(t)\boldsymbol{P}_i^{o*}(t-1)\boldsymbol{H}_i^{o^T}(t)\right]^{-1} \tag{30}$$

$$\boldsymbol{P}_i^o(t) = \boldsymbol{P}_i^{o*}(t-1) - \boldsymbol{K}_i^o(t)\boldsymbol{H}_i^o(t)\boldsymbol{P}_i^{o*}(t-1) \tag{31}$$

$$\begin{aligned} \hat{\boldsymbol{w}}_i^o(t) = &\hat{\boldsymbol{w}}_i^o(t-1) - \alpha \boldsymbol{P}_i^o(t)\hat{\boldsymbol{w}}_i^o(t-1) \\ &+ \boldsymbol{K}_i^o(t)[d_i(t) - h_i(\hat{\boldsymbol{w}}(t-1), \boldsymbol{x}(t))], \end{aligned} \tag{32}$$

where

$$\boldsymbol{H}_i^o(\tau) = \left[\left.\frac{\partial h_i(\boldsymbol{w}, \boldsymbol{x}(\tau))}{\partial \boldsymbol{w}_i^o}\right|_{\boldsymbol{w}=\hat{\boldsymbol{w}}(\tau-1)}\right]^T, \tag{33}$$

is the $1 \times (n_h + 1)$ decoupled gradient matrix, $\boldsymbol{K}_i^o(t)$ is the $(n_h + 1) \times 1$ decoupled Kalman gain, and $\boldsymbol{P}_i^o(t)$ is the $(n_h + 1) \times (n_h + 1)$ decoupled error covariance matrix.

Similarly, for the *j*th hidden neuron, it is associated with a decoupled weight vector $\boldsymbol{w}_j^{\text{in}}$. The energy function of this decoupled weight vector $\boldsymbol{w}_j^{\text{in}}$ is given by

$$\begin{aligned} &E(\boldsymbol{w}_j^{\text{in}}) \\ &= \sum_{\tau=1}^{t}\left[[\boldsymbol{d}(\tau) - \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(\tau))]^T[\boldsymbol{d}(\tau) - \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(\tau))] + \alpha \boldsymbol{w}_j^{\text{in}^T}\boldsymbol{w}_j^{\text{in}}\right] \\ &\quad + \left[\boldsymbol{w}_j^{\text{in}} - \hat{\boldsymbol{w}}_j^{\text{in}}(0)\right]^T \boldsymbol{P}_j^{\text{in}^{-1}}(0)\left[\boldsymbol{w}_j^{\text{in}} - \hat{\boldsymbol{w}}_j^{\text{in}}(0)\right]. \end{aligned} \tag{34}$$

With the energy function, the recursive equations are given by

$$\boldsymbol{P}_j^{\text{in}*}(t-1) = \left[\boldsymbol{I}_{(n_{\text{in}}+1)\times(n_{\text{in}}+1)} + \alpha \boldsymbol{P}_j^{\text{in}}(t-1)\right]^{-1}\boldsymbol{P}_j^{\text{in}}(t-1) \tag{35}$$

$$\begin{aligned} \boldsymbol{K}_j^{\text{in}}(t) = &\boldsymbol{P}_j^{\text{in}*}(t-1)\boldsymbol{H}_j^{\text{in}^T}(t) \\ &\times \left[\boldsymbol{I}_{(n_o+1)\times(n_o+1)} + \boldsymbol{H}_j^{\text{in}}(t)\boldsymbol{P}_j^{\text{in}*}(t-1)\boldsymbol{H}_j^{\text{in}^T}(t)\right]^{-1} \end{aligned} \tag{36}$$

$$\boldsymbol{P}_j^{\text{in}}(t) = \boldsymbol{P}_j^{\text{in}*}(t-1) - \boldsymbol{K}_j^{\text{in}}(t)\boldsymbol{H}_j^{\text{in}}(t)\boldsymbol{P}_j^{\text{in}*}(t-1) \tag{37}$$

$$\begin{aligned} \hat{\boldsymbol{w}}_j^{\text{in}}(t) = &\hat{\boldsymbol{w}}_j^{\text{in}}(t-1) - \alpha \boldsymbol{P}_j^{\text{in}}(t)\hat{\boldsymbol{w}}_j^{\text{in}}(t-1) \\ &+ \boldsymbol{K}_j^{\text{in}}(t)[\boldsymbol{d}(t) - \boldsymbol{h}(\hat{\boldsymbol{w}}(t-1), \boldsymbol{x}(t))], \end{aligned} \tag{38}$$

where

$$\boldsymbol{H}_j^{\text{in}}(\tau) = \left[\left.\frac{\partial \boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x}(\tau))}{\partial \boldsymbol{w}_j^{\text{in}}}\right|_{\boldsymbol{w}=\hat{\boldsymbol{w}}(\tau-1)}\right]^T, \tag{39}$$

is the $n_o \times (n_{\text{in}} + 1)$ decoupled gradient matrix, $\boldsymbol{K}_j^{\text{in}}(t)$ is the $(n_{\text{in}} + 1) \times n_o$ decoupled Kalman gain, and $\boldsymbol{P}_j^{\text{in}}(t)$ is the $(n_{\text{in}} + 1) \times (n_{\text{in}} + 1)$ decoupled error covariance matrix.

The training process of the decoupled TWDRLS algorithm is as follows. We first train the output decoupled weight vectors $\boldsymbol{w}_i^o$'s. Afterwards, we update the hidden decoupled weight vectors $\boldsymbol{w}_j^{\text{in}}$'s. At each training stage, only the concerned weight vector is updated and all other weights remain unchanged.

3.2 Complexity

In the global TWDRLS, the complexity mainly comes from computing the inverse of the *M*-dimensional matrix $(I_{M\times M} + \alpha P(t-1))$. This complexity is equal to $O(M^3)$. So, the computational complexity is equal to

$$TCC_{\text{global}} = O(M^3) = O\Big(\left(n_o(n_{h+1}) + n_h(n_{\text{in}} + 1)\right)^3\Big).$$

Since the size of the matrix is $M \times M$, the space complexity (storage requirement) is equal to

$$TCS_{\text{global}} = O(M^2) = O\Big(\left(n_o(n_{h+1}) + n_h(n_{\text{in}} + 1)\right)^2\Big).$$

[1] It should be noticed that such a technique is usually used in many numerical methods [15, 23, 24]. That means, at each training iteration, we update each decoupled weight vector separately.

From (29), for each output neuron, the computational cost of the decoupled TWDRLS algorithm mainly comes from the inversion of an $(n_h + 1) \times (n_h + 1)$ matrix. In this way, the computational complexity for each output neuron is $O((n_h + 1)^3)$ and the corresponding space complexity is equal to $O((n_h + 1)^2)$. From (35), for each hidden neuron, the computational cost of the decoupled TWDRLS algorithm mainly comes from the inversion of an $(n_{\text{in}} + 1) \times (n_{\text{in}} + 1)$ matrix. In this way, the computational complexity for each hidden neuron is $O((n_{\text{in}} + 1)^3)$ and the corresponding space complexity is equal to $O((n_{\text{in}} + 1)^2)$.

Hence, the total computational complexity of the decoupled TWDRLS algorithm is equal to

$$TCC_{\text{decouple}} = O\left(n_o(n_h + 1)^3 + n_h(n_{\text{in}} + 1)^3\right)$$

and the space complexity (storage requirement) is equal to

$$TCS_{\text{decouple}} = O\left(n_o(n_h + 1)^2 + n_h(n_{\text{in}} + 1)^2\right).$$

They are much smaller than the computational and space complexities of the global case. Some examples related to the complexity issue will be given in the next section.

4 Computer simulations

The proposed decoupled TWDRLS algorithm is applied to two problems: the generalized XOR problem and the sunspot data prediction problem. Its performance is compared with that of the global version. The first problem is a typical nonlinear classification problem while the second one is a standard nonlinear time series prediction problem. The initial weights are small zero-mean independent identically distributed Gaussian random variables. The activation function for hidden neurons is the hyperbolic tangent function. Since the generalized XOR problem is a classification problem, the output neuron of the generalized XOR problem is with the hyperbolic tangent activation function. Since the sunspot data prediction problem is a regression problem, the output neuron of the sunspot data prediction problem is with the linear activation function. The training for each problem is performed 10 times with different random initial weights.

4.1 Generalized XOR problem

The generalized XOR problem is formulated as $d = sign(x_1x_2)$ with inputs in the range $[-1, 1]$. The desired output is either -1 (corresponding to logical zero) or 1 (corresponding to logical one). The network has two input neurons, ten hidden neurons, and one output neuron. As a result, there are 41 weights in the network. The training set and test set, shown in Fig. 1, contain 50 and 2,000 samples, respectively. The total number of training cycles is set to 200. It is because after 200 training cycles, the decreasing rate of training errors is very slow. In each cycle, 50 randomly selected training samples from the training set are fed to the network one by one.

Since the generalized XOR problem is a classification problem, the criterion used to evaluate the model performance is the false rate (misclassification rate). A test pattern is misclassified when the sign of the network output is not the same as that of the desired one. Figure 2 summarizes the average test set false rates in the ten runs. The average test set false rates obtained by global and decouple TWDRLS algorithms are usually lower than those obtained by the standard RLS and decouple RLS algorithms over a wide range of regularization parameters. This means that both global and decouple TWDRLS algorithms can improve the generalization ability. In terms of average false rate, the performance of the decouple TWDRLS

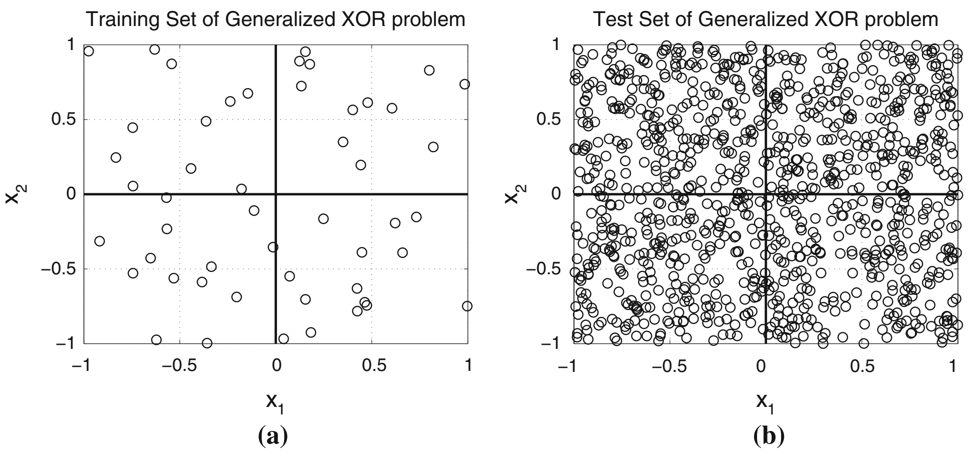

Fig. 1 Training and test samples for the generalized XOR problems. **a** Training samples. **b** Test samples

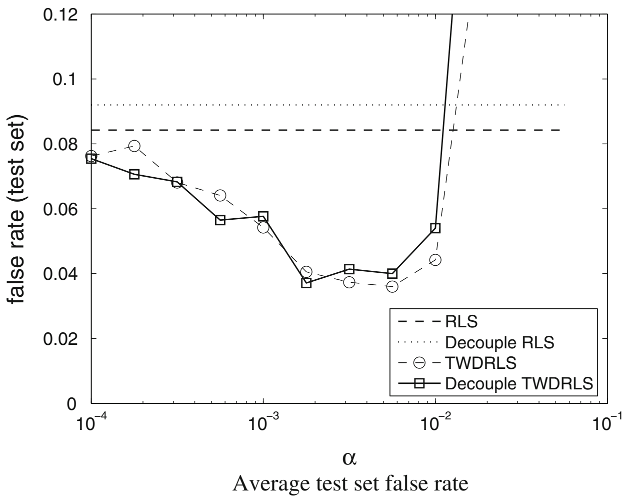

Fig. 2 Average test set false rate of 10 runs for the generalized XOR problem

algorithm is quite similar to that of the global ones. The computational and space complexities for global and decouple algorithms are listed in Table 1. From Fig. 2 and Table 1, we can conclude that the performance of the decouple TWDRLS algorithm is comparable to that of the global ones and that its time and space complexities are much smaller.

Table 1 Computational and space complexities of the global and decouple TWDRLS algorithms for the generalized XOR problem

Algorithm	Computational complexity	Space complexity
Global	$O(6.89 \times 10^4)$	$O(1.68 \times 10^3)$
Decouple	$O(1.60 \times 10^3)$	$O(2.21 \times 10^2)$

The decision boundaries obtained from typical trained networks are plotted in Fig. 3. From Figs. 1 and 3, the decision boundaries obtained from the trained networks with TWDRLS algorithms are closer to the ideal ones. Also, the performance of decouple TWDRLS is very close to that of the global TWDRLS algorithm.

From Fig. 2, the average test set false rate first decreases with the regularization parameter α and then increases with it. This shows that a proper selection of α indeed improves the generalization ability of the network. On the other hand, we observe that the test set false rate becomes very high when the decay parameter α is very large. This is due to the fact that when the decay parameter is very large, the weight decay effect is very

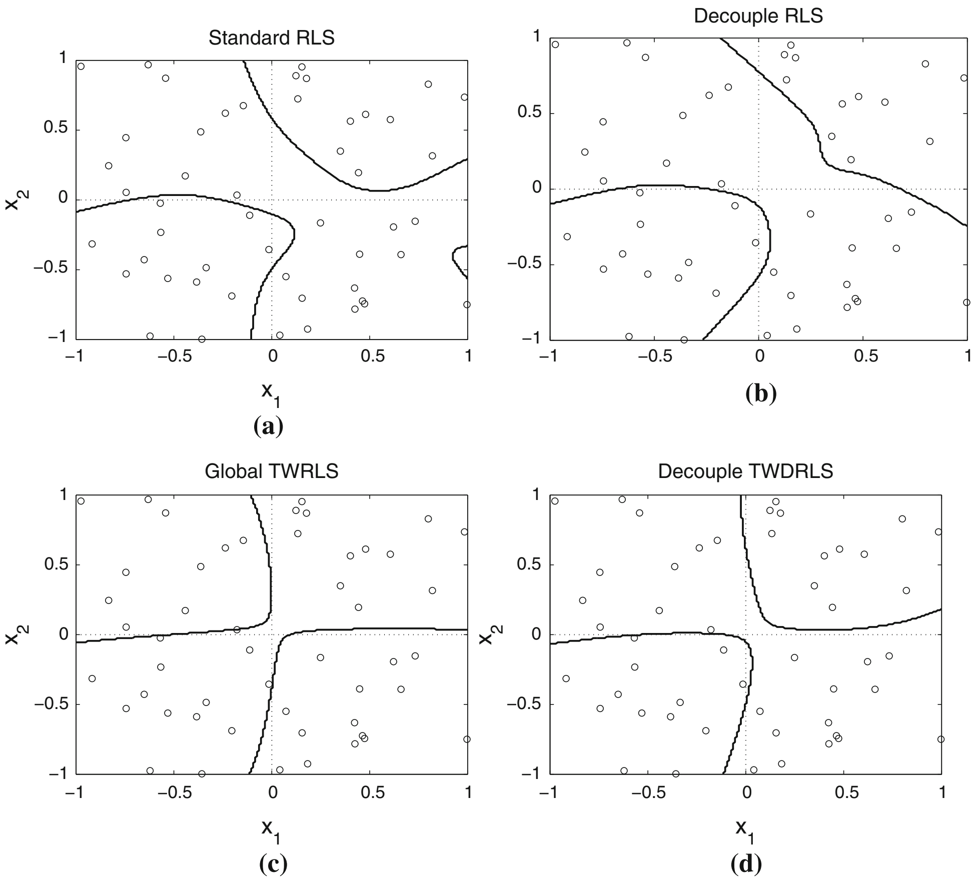

Fig. 3 Decision boundaries of various trained networks for the generalized XOR problem. Note that when $\alpha = 0$, the TWDRLS is identical to RLS. **a** RLS. **b** Decouple RLS. **c** Global TWDRLS, $\alpha = 0.00562$. **d** Decouple TWDRLS, $\alpha = 0.00178$

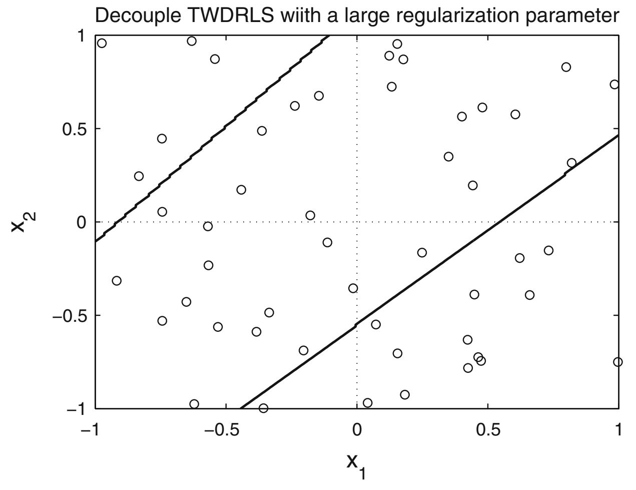

Fig. 4 Decision boundary of a trained network (decouple TWDRLS) with too large regularization parameter, where $\alpha = 0.0178$. Since the regularization parameter is too large, the trained network cannot form a good decision boundary

substantial and the trained network cannot learn the target function. In order to further illustrate this, we plot in Fig. 4 the decision boundary of the network trained with $\alpha = 0.0178$. The figure shows that the decision boundary is quite far from the ideal one. This is because when the value of α is too large, the weight decay effect is too strong and then the trained network cannot capture the desired decision boundary.

4.2 Sunspot data prediction

The sunspot data from 1700 to 1979, shown in Fig. 5, are taken as the training and the test sets. Following the common practice, we divide the data into a training set (1700–1920) and two test sets, namely, Test-set 1 (1921–1955) and Test-set 2 (1956–1979). The sunspot series is rather non-stationary and Test-set 2 is atypical.

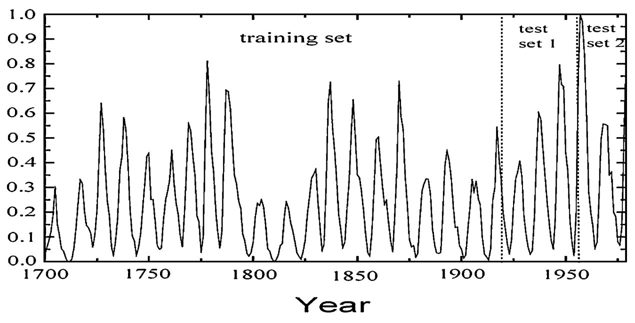

Fig. 5 Sunspot data

In the simulation, we assume that the series is generated from the following auto-regressive model, given by

$$d(t) = \varphi(d(t-1), \ldots, d(t-12)) + \epsilon(t) \tag{40}$$

where $\epsilon(t)$ is noise and $\varphi(\cdot, \ldots, \cdot)$ is an unknown nonlinear function. A network with 12 input neurons, 8 hidden neurons, and one output neuron is used for approximating $\varphi(\cdot, \ldots, \cdot)$. There are 113 weights in the MFNN model. The total number of training cycles is equal to 200. As this is a time series problem, the criterion to evaluate the model performance is the mean squared error (MSE) of the test set.

Figure 6 summarizes the average MSE in the 10 runs. From Fig. 6, over a wide range of the regularization parameter α, both global and decouple TWDRLS algorithms can greatly improve the generalization ability of the trained networks, especially for Test-set 2 that is quite different from the training set. However, the test MSE becomes very large at large values of α. This is because at large value of α, the weight decay effect is too strong and then the network cannot learn the target function. In most cases, the performance of the decouple training is comparable to that of the global ones. Also, Table 2 shows that those complexities of the decouple training are much smaller than those of the global one.

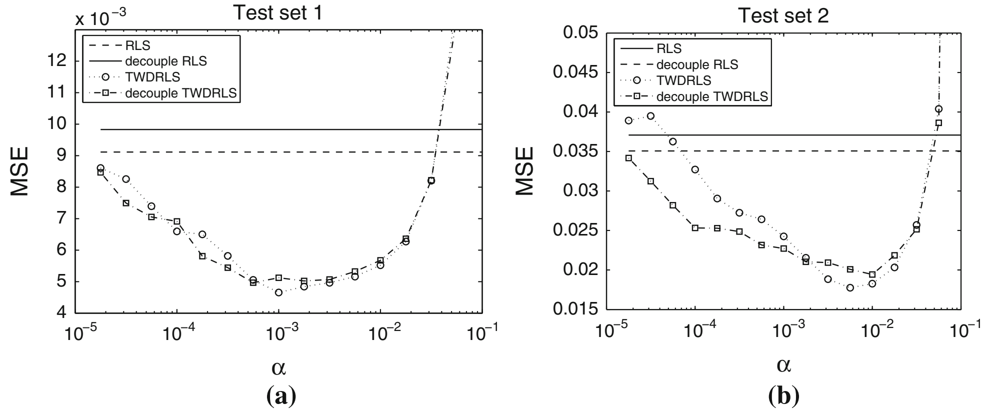

Fig. 6 MSE of networks trained by global and decouple TWDRLS algorithms. Note that when $\alpha = 0$, the TWDRLS is identical to RLS. **a** Test-set 1 average MSE. **b** Test-set 2 average MSE

Table 2 Computational and space complexities of the global and decouple TWDRLS algorithms for the sunspot data prediction problem

Algorithm	Computational complexity	Space complexity
Global	$O(1.44 \times 10^6)$	$O(1.28 \times 10^4)$
Decouple	$O(1.83 \times 10^4)$	$O(1.43 \times 10^3)$

5 Conclusion

We have investigated the problem of training the MFNN model using the TWDRLS algorithms. We derive a set of concise equations for the decouple TWDRLS algorithm. Computer simulations indicate that both decouple and global TWDRLS algorithms can improve the generation ability of MFNNs. The performance of the decouple TWDRLS algorithm is comparable to that of the global ones. However, when the decouple approach is used, the computational complexity and the storage requirement are greatly reduced. In the decoupled version, each neuron has its own set of RLS equations. Hence, the decoupled version is suitable for the parallel computing [25, 26]. Hence, one of the future works is to develop a parallel implementation of the decoupled version, in which each processor is used for the computation of one set of RLS equations.

Acknowledgments The work was supported by a research grant from City University of Hong Kong (7002701).

References

1. Leung CS, Wong KW, Sum PF, Chan LW (1996) On-line training and pruning for RLS algorithms. Electron Lett 32(23):2152–2153
2. Leung CS, Wong KW, Sum PF, Chan LW (2001) A pruning method for the recursive least squared algorithm. Neural Netw 14(2):147–174
3. Leung CS, Sum J, Young GH, Kan WK (1999) On the Kalman filtering method in neural networks training and pruning. IEEE Trans Neural Netw 10(1):161–165
4. Scalero R, Tepedelelenlioglu N (1992) Fast new algorithm for training feedforward neural networks. IEEE Trans Signal Process 40(1):202–210
5. Shah S, Palmieri F, Datum F (1992) Optimal filtering algorithms for fast learning in feedforward neural networks. Neural Netw 5(5):779–787
6. Rumelhart D, Hinton G, Williams R (1986) Learning internal representations by error propagation, pp 318–362
7. Moody JE (Sep. 1991) Note on generalization, regularization, and architecture selection in nonlinear learning systems. In: Proceedings first IEEE-SP workshop on neural networks for signal processing, pp 1–10
8. Krogh A, Hertz JA (1992) A simple weight decay can improve generalization. In: Adv Neural Inf Process Syst 4, [NIPS conference]. Morgan Kaufmann, pp 950–957
9. Chen S, Hong X, Harris C, Sharkey P (2004) Sparse modelling using orthogonal forward regression with press statistic and regularization. IEEE Trans Syst Man Cybern B 34(2):898–911
10. Sum J, Leung CS, Ho KI-J (2009) On objective function, regularizer, and prediction error of a learning algorithm for dealing with multiplicative weight noise. IEEE Trans Neural Netw 20(1):124–138
11. Bernier JL, Ortega J, Rojas I, Ros E, Prieto A (2000) Obtaining fault tolerant multilayer perceptrons using an explicit regularization. Neural Process Lett 12(2):107–113
12. Leung CS, Tsoi AC, Chan LW (2001) Two regularizers for recursive least square algorithms in feedforward multilayered neural networks. IEEE Trans Neural Netw 12(6):1314–1332
13. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4:251–257
14. Mosca E (1995) Optimal predictive and adaptive control. Prentice-Hall, Englewood Cliffs, NJ
15. Haykin S (1991) Adaptive filter theory. Prentice-Hall, Englewood Cliffs, NJ
16. Mackay D (1992) Bayesian interpolation. Neural Comput Appl 4:415–447
17. Mackay D (1992) A practical bayesian framework for backpropagation networks. Neural Comput Appl 4:448–472
18. Moody JE, Hanson SJ, Lippmann SJ (1992) The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. Adv Neural Inf Process Syst 4 [NIPS Conference]: 847–854
19. Rögnvaldsson TS (1998) A simple trick for estimating the weight decay parameter. In Neural networks: tricks of the trade, this book is an outgrowth of a 1996 NIPS workshop. Springer, London, pp 71–92
20. Sugiyama M, Ogawa H (2002) Optimal design of regularization term and regularization parameter by subspace information criterion. Neural Netw 15(3):349–361
21. Guo P (2002) Studies of model selection and regularization for generalization in neural networks with applications. Ph.D. dissertation, The Chinese University of Hong Kong, Hong Kong, supervisor-Michael R. Lyu
22. Guo P, Lyu M, Chen C (2003) Regularization parameter estimation for feedforward neural networks. IEEE Trans Syst Man Cybern B 33(1):35–44
23. Hager WW (1989) Applied numerical linear algebra. Prentice-Hall, Englewood Cliffs, NJ
24. Li S, Wunsch DC, O'Hair E, Giesselmann MG (2002) Extended Kalman filter training of neural networks on a simd parallel machine. J Parallel Distrib Comput 62(4):544–562
25. Xiao Y, Leung C-S, Ho T-Y, Lam P-M (2011) A gpu implementation for lbg and som training. Neural Comput Appl 20(7):1035–1042. doi:10.1007/s00521-010-0403-7
26. Ho T-Y, Lam P-M, Leung C-S (2008) Parallelization of cellular neural networks on gpu. Pattern Recogn Lett 41(8):2684–2692