2

Neural network Wiener models

2.1 Introduction

This chapter introduces different structures of neural network Wiener models and shows how their weights can be adjusted, based on a set of system input-output data, with gradient learning algorithms. The term 'neural network Wiener models' refers to models composed of a linear dynamic model followed by a nonlinear multilayer perceptron model. Both the SISO nad MISO Wiener models in their two basic configurations known as a series-parallel and a parallel model are considered. In series-parallel Wiener models, another multilayer perceptron is used to model the inverse nonlinear element. For neural network Wiener models, four different rules for the calculation of the gradient or the approximate gradient are derived and presented in a unified framework. In series-parallel models, represented by feedforward neural networks, the calculation of the gradient can be carried out with the backpropagation method (BPS). Three other methods, i.e., backpropagation for parallel models (BPP), the sensitivity method (SM), and truncated backpropagation through time (BPTT) are used to calculate the gradient or the approximate gradient in parallel models. For the BPTT method, it is shown that the accuracy of gradient approximation depends on both the number of unfolded time steps and impulse response functions of the linear dynamic model and its sensitivity models. Computational complexity of the algorithms is also analyzed and expressed in terms of the orders of polynomials describing the linear dynamic model, the number of nonlinear nodes, and the number of unfolded time steps. Having the gradient calculated, different gradient-based algorithms such as the steepest descent, quasi-Newton (or variable metric), and conjugate gradient can be applied easily.

All the learning algorithms, discussed in this chapter, are one-step identification procedures, i.e., they allow one to identify both the nonlinear element and the linear dynamic system simultaneously. The sequential (on-line) mode of the algorithms makes them also suitable for the identification of systems with slowly varying parameters. An important advantage of the parallel Wie-

A. Janczak: Identification of Nonlinear Systems, LNCIS 310, pp. 31–75, 2005.

ner model is that it does not require the assumption of invertibility of the nonlinear element. Due to output error formulation of the prediction error, parallel Wiener models are also useful for the identification of systems disturbed additively by the white output noise.
The chapter is organized as follows: The identification problem is formulated in Section 2.2. Then the SISO and MISO series-parallel and parallel neural network Wiener models are introduced in Section 2.3. Section 2.4 contains details of different gradient calculation algorithms and an analysis of gradient computation accuracy with the truncated BPTT method. Simulation results and a real data example of a laboratory two-tank system are shown in Sections 2.5 and 2.6 to compare the convergence rates of the algorithms. The recursive prediction error learning algorithm is derived in Section 2.7. Finally, Section 2.8 summarizes the essential results.

2.2 Problem formulation

Consider a two-block structure composed of a linear dynamic system and a static nonlinear element in a cascade, referred to as the SISO Wiener system – Fig. 2.1. The output $y(n)$ to the input $u(n)$ at the time n is

$$y(n) = f\left(\frac{B(q^{-1})}{A(q^{-1})}u(n)\right) + \varepsilon(n), \tag{2.1}$$

where

$$A(q^{-1}) = 1 + a_1 q^{-1} + \cdots + a_{na} q^{-na}, \tag{2.2}$$

$$B(q^{-1}) = b_1 q^{-1} + \cdots + b_{nb} q^{-nb}, \tag{2.3}$$

and q^{-1} is the backward shift operator, with the properties that $q^{-m}y(n) = y(n-m)$ and $q^{-m}f\big(s(n)\big) = f\big(s(n-m)\big)$, $f(\cdot)$ is the characteristic of the nonlinear element, $a_1, \ldots, a_{na}$, $b_1, \ldots, b_{nb}$ are the unknown parameters of the linear dynamic system, and $\varepsilon(n)$ is the system output disturbance. Let us assume that:
Assumption 2.1. The function $f(\cdot)$ is continuous.
Assumption 2.2. The linear dynamic system is casual and asymptotically stable.
Assumption 2.3. The polynomial orders na and nb are known.
The identification problem can be formulated as follows: Given the sequence of the system input and output measurements $\{u(n), y(n)\}, n = 1, \ldots, N$, estimate the parameters of the linear dynamic system and the characteristic of the nonlinear element minimizing the following global cost function:

$$J = \frac{1}{2}\sum_{n=1}^{N}\big(y(n) - \hat{y}(n)\big)^2, \tag{2.4}$$

where $\hat{y}(n)$ is the output of the neural network Wiener model.

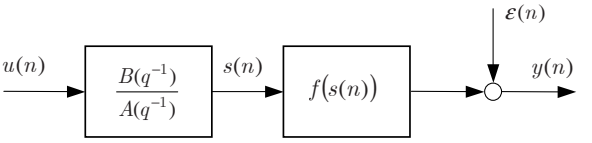

Fig. 2.1. Wiener system

There are two basic approaches to find the minimum of the global cost function (2.4). The first method is called the sequential mode or pattern learning as it uses pattern-by-pattern updating of model parameters, changing their values by an amount proportional to the negative gradient of the local cost function

$$J(n) = \frac{1}{2}\big(y(n) - \hat{y}(n)\big)^2. \tag{2.5}$$

This results in the following rule for the adaptation of model parameters:

$$\mathbf{w}(n) = \mathbf{w}(n-1) - \eta \frac{\partial J(n)}{\partial \mathbf{w}(n-1)}, \tag{2.6}$$

$$\frac{\partial J(n)}{\partial \mathbf{w}(n-1)} = -\big(y(n) - \hat{y}(n)\big) \frac{\partial \hat{y}(n)}{\partial \mathbf{w}(n-1)}, \tag{2.7}$$

where $\mathbf{w}(n)$ is the weight vector containing all model parameters at the time n, and $\eta > 0$ is the learning rate. It has been shown that such a learning procedure minimizes the global cost function J provided that the learning rate η is sufficiently small [167].

The other approach is called batch learning as it uses the whole set of input-output data $\{u(n), y(n)\}, n = 1, \ldots, N$, to update model parameters. In this technique, the global cost function J is minimized iteratively in such a way that, at each iteration, the parameter changes over all training patterns are accumulated before the parameters are actually changed.

The gradient learning algorithm in its basic version has some serious drawbacks. The fixed learning rate η may be chosen too large leading to unnecessary oscillations or even the divergence of the learning process. On the other hand, at too small η, the learning process may become extremely slow. Many different modifications of the gradient algorithm exist which improve and speed up the learning process considerably. Adding a momentum term is one simple way to improve the learning. Including the momentum term smoothes weight changes making it possible to increase the learning rate η. This may not only speed up the learning process but may also prevent it from getting trapped in a shallow local minimum. Also, applying diverse learning rates for different weights, neurons, or layers may be a useful tool to improve the learning process. Other effective modifications of the gradient algorithm are based on the adaptation of the learning rate η according to some heuristic rules [68].

It is well known that parallel models can loose their stability during the learning process even if the identified system is stable [121]. A common way to avoid this is to keep the learning rate η small. To preserve the stability of Wiener models, some other heuristic rules can be applied easily. For example, asymptotic stability of the model can be tested after each parameter update and if the model becomes unstable, the unstable poles can be moved back into the unit circle by scaling. This requires recalculation and scaling the parameters of the linear dynamic model to keep the steady-state gain of the model unchanged. Another simple approach, which can be applied in the sequential mode, is based on the idea of a trial update of parameters, testing the stability of the model, and performing an actual update only if the obtained model is asymptotically stable; otherwise the parameters are not updated and the next training pattern is processed.

2.3 Series-parallel and parallel neural network Wiener models

In analogy with linear dynamic models, neural network Wiener models can be used in two basic configurations known as series parallel and parallel models. Neural network architectures of both of these models are introduced and discussed below.

2.3.1 SISO Wiener models

To introduce series-parallel Wiener models, it is necessary to assume that the nonlinear function $f(\cdot)$ is invertible. A series-parallel neural network Wiener model, shown in Fig. 2.2, is composed of a multilayer perceptron model of the inverse nonlinear element, a linear node with two tapped delay lines, used as a model of the linear dynamic system, and another multilayer perceptron used as a model of the nonlinear element. The input to the model are the system input $u(n)$ and system output $y(n)$, and the model is of the feedforward type. Multilayer perceptrons are universal approximators. This means that the multilayer perceptron with at least one hidden layer can approximate any smooth function to an arbitrary degree of accuracy as the number of hidden layer neurons increases [37]. Multilayer perceptrons with one hidden layer are most common. Therefore, we assume that both the model of the nonlinear element and the inverse model of the nonlinear element have the same architecture, and contain one hidden layer of M nonlinear processing elements, see Fig. 2.3 for the nonlinear element model and Fig 2.4 for the inverse nonlinear element model. The output $\hat{y}(n)$ of the series-parallel neural network Wiener model is

$$\hat{y}(n) = \hat{f}\big(\hat{s}(n),\mathbf{w}\big), \tag{2.8}$$

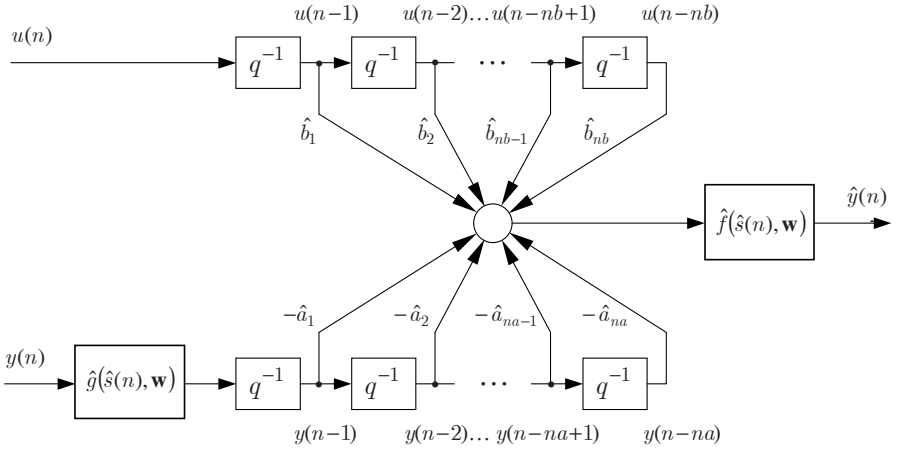

Fig. 2.2. Series-parallel SISO neural network Wiener model

with

$$\hat{s}(n) = -\sum_{m=1}^{na} \hat{a}_m \hat{g}\big(y(n-m), \mathbf{v}\big) + \sum_{m=1}^{nb} \hat{b}_m u(n-m), \tag{2.9}$$

$$\hat{f}\big(\hat{s}(n), \mathbf{w}\big) = \sum_{j=1}^{M} w_{1j}^{(2)} \varphi\big(x_j(n)\big) + w_{10}^{(2)}, \tag{2.10}$$

$$x_j(n) = w_{j1}^{(1)} \hat{s}(n) + w_{j0}^{(1)}, \tag{2.11}$$

$$\hat{g}\big(y(n), \mathbf{v}\big) = \sum_{j=1}^{M} v_{1j}^{(2)} \varphi\big(z_j(n)\big) + v_{10}^{(2)}, \tag{2.12}$$

$$z_j(n) = v_{j1}^{(1)} y(n) + v_{j0}^{(1)}, \tag{2.13}$$

where the function $\hat{f}(\cdot)$ describes the nonlinear element model, the function $\hat{g}(\cdot)$ describes the inverse nonlinear element model, $\varphi(\cdot)$ is the activation function, $\hat{a}_1, \ldots, \hat{a}_{na}$, $\hat{b}_1, \ldots, \hat{b}_{nb}$ are the parameters of the linear dynamic model, $\mathbf{w} = [w_{10}^{(1)} \ldots \; w_{M1}^{(1)} \; w_{10}^{(2)} \; \ldots \; w_{1M}^{(2)}]^T$ is the parameter (weight) vector of the nonlinear element model, and $\mathbf{v} = [v_{10}^{(1)} \; \ldots \; v_{M1}^{(1)} \; v_{10}^{(2)} \; \ldots \; v_{1M}^{(2)}]^T$ is the parameter vector of the inverse nonlinear element model. Note that $\hat{g}(\cdot)$ does not denote the inverse of $\hat{f}(\cdot)$ but describes the inverse nonlinear element. The architecture of the parallel model (Fig. 2.5), which does not contain any inverse model, is even simpler in comparison with that of the series-parallel one. The parallel model is of the recurrent type as it contains a single feedback

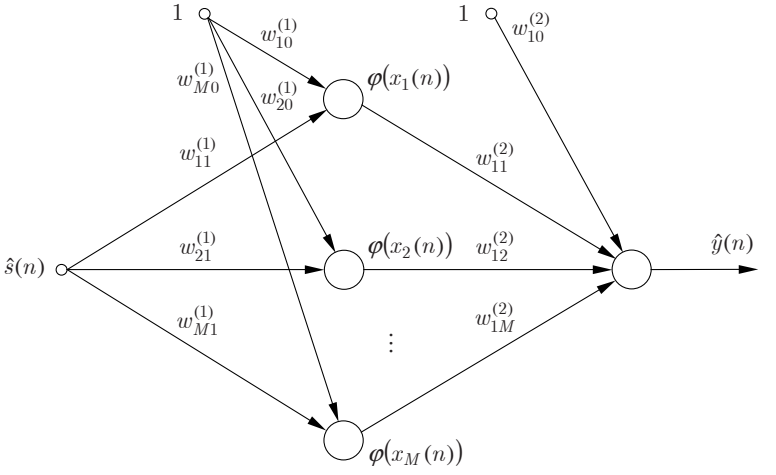

Fig. 2.3. Neural network model of the SISO nonlinear element

connection and $u(n)$ is the only input to the model. The output $\hat{y}(n)$ of the parallel Wiener model is given by

$$\hat{y}(n) = \hat{f}\big(\hat{s}(n),\mathbf{w}\big), \tag{2.14}$$

with $\hat{f}(\cdot)$ defined by (2.10) and (2.11), and where the output of the linear dynamic model is given by the following difference equation:

$$\hat{s}(n) = -\sum_{m=1}^{na} \hat{a}_m \hat{s}(n-m) + \sum_{m=1}^{nb} \hat{b}_m u(n-m). \tag{2.15}$$

Using the backward shift operator notation, (2.15) can be written as

$$\hat{s}(n) = \big[1 - \hat{A}(q^{-1})\big]\hat{s}(n) + \hat{B}(q^{-1})u(n), \tag{2.16}$$

where $\hat{A}(q^{-1}) = 1+\hat{a}_1 q^{-1}+\cdots+\hat{a}_{na}q^{-na}$ and $\hat{B}(q^{-1}) = \hat{b}_1 q^{-1}+\cdots+\hat{b}_{nb}q^{-nb}$.

Note that the characterization of the Wiener system is not unique as the linear dynamic system and the nonlinear element are connected in series. In other words, Wiener systems described by $\big(B(q^{-1})/A(q^{-1})\big)/\alpha$ and $f\big(\alpha s(n)\big)$ reveal the same input-output behavior for any $\alpha \neq 0$. Therefore, to obtain a unique characterization of the neural network model, either the linear dynamic model or the nonlinear element model should be normalized. For example, to normalize the gain of the linear dynamic model to 1, the model weights are scaled as follows: $\hat{b}_k = \tilde{b}_k/\alpha$, $k = 1,\ldots,nb$, $w_{j1}^{(1)} = \alpha\tilde{w}_{j1}^{(1)}$, $j = 1,\ldots,M$, where $\tilde{b}_k$, $\tilde{w}_{j1}^{(1)}$ denote the parameters of the unnormalized model, and $\alpha = \tilde{B}(1)/\tilde{A}(1)$ is the linear dynamic model gain.

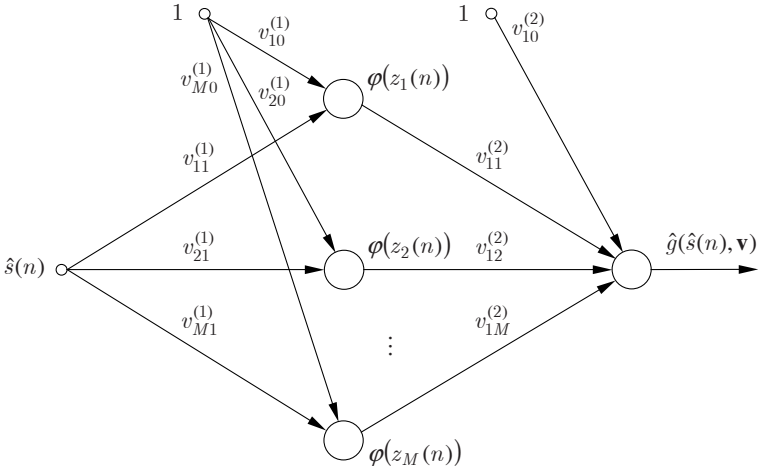

Fig. 2.4. Neural network model of the SISO inverse nonlinear element

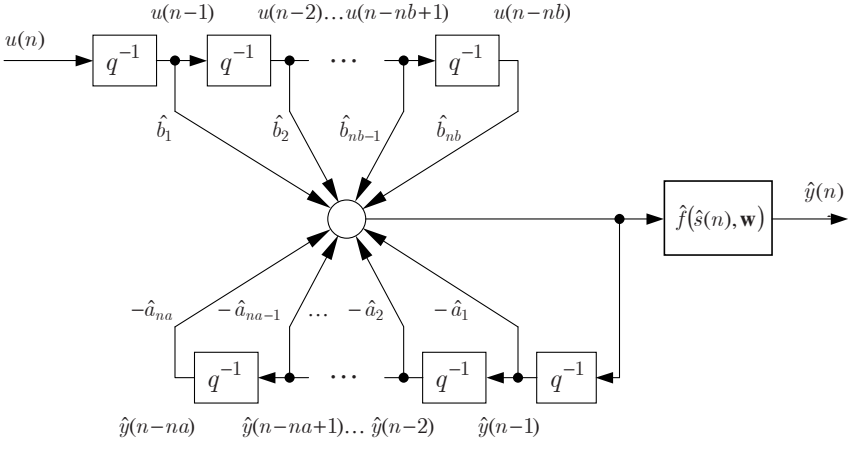

Fig. 2.5. Parallel SISO neural network Wiener model

2.3.2 MIMO Wiener models

Consider a parallel MIMO neural network Wiener model with nu inputs, ny outputs, ns outputs of the MIMO linear dynamic model, and M nonlinear nodes in the hidden layer. Assume that a single hidden layer multilayer perceptron with ns inputs and ny outputs, containing M nonlinear nodes in its hidden layer, is used as a model of the nonlinear element – Fig 2.6. The output $\hat{\mathbf{y}}(n)$ of the model at the time n is

$$\hat{\mathbf{y}}(n) = \hat{\mathbf{f}}\big(\hat{\mathbf{s}}(n),\mathbf{W}\big), \tag{2.17}$$

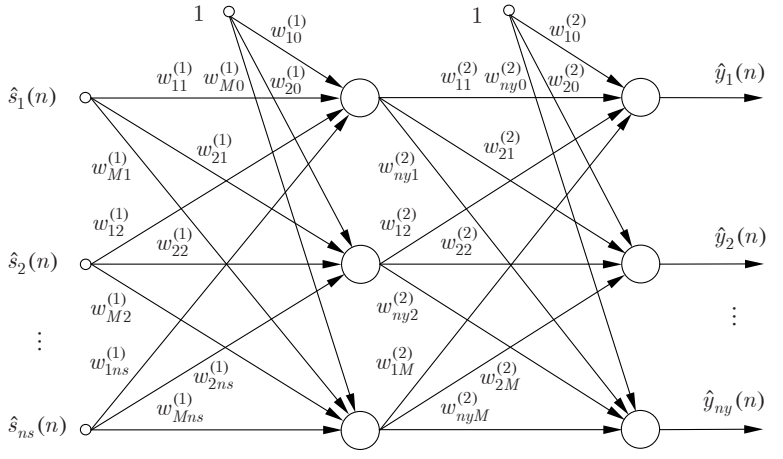

Fig. 2.6. Neural network model of the MIMO nonlinear element

where

$$\hat{\mathbf{y}}(n) = \big[\hat{y}_1(n) \dots \hat{y}_{ny}(n)\big]^T, \tag{2.18}$$

$$\hat{\mathbf{f}}(\hat{\mathbf{s}}(n), \mathbf{W}) = \big[\hat{f}_1\big(\hat{\mathbf{s}}(n), \mathbf{w}_1\big) \dots \hat{f}_{ny}\big(\hat{\mathbf{s}}(n), \mathbf{w}_{ny}\big)\big]^T, \tag{2.19}$$

$$\hat{\mathbf{s}}(n) = \big[\hat{s}_1(n) \dots \hat{s}_{ns}(n)\big]^T, \tag{2.20}$$

$$\mathbf{W} = \big[w_{10}^{(1)} \dots w_{Mns}^{(1)}\ w_{10}^{(2)} \dots w_{nyM}^{(2)}\big]^T, \tag{2.21}$$

$$\mathbf{w}_t = \big[w_{10}^{(1)} \dots w_{Mns}^{(1)}\ w_{t0}^{(2)} \dots w_{tM}^{(2)}\big]^T, \tag{2.22}$$

where $t = 1, \dots, ny$. Then the tth output of the nonlinear element model is

$$\hat{y}_t(n) = \hat{f}_t\big(\hat{\mathbf{s}}(n), \mathbf{w}_t\big) = \sum_{j=1}^{M} w_{tj}^{(2)} \varphi\big(x_j(n)\big) + w_{t0}^{(2)}, \tag{2.23}$$

$$x_j(n) = \sum_{i=1}^{ns} w_{ji}^{(1)} \hat{s}_i(n) + w_{j0}^{(1)}. \tag{2.24}$$

The output $\hat{\mathbf{s}}(n)$ of the MIMO dynamic model can be expressed as

$$\hat{\mathbf{s}}(n) = -\sum_{m=1}^{na} \hat{\mathbf{A}}^{(m)} \hat{\mathbf{s}}(n-m) + \sum_{m=1}^{nb} \hat{\mathbf{B}}^{(m)} \mathbf{u}(n-m), \tag{2.25}$$

where

$$\mathbf{u}(n) = \big[u_1(n) \dots u_{nu}(n)\big]^T, \tag{2.26}$$

$$\hat{\mathbf{A}}^{(m)} \in \mathbb{R}^{ns \times ns}, \tag{2.27}$$

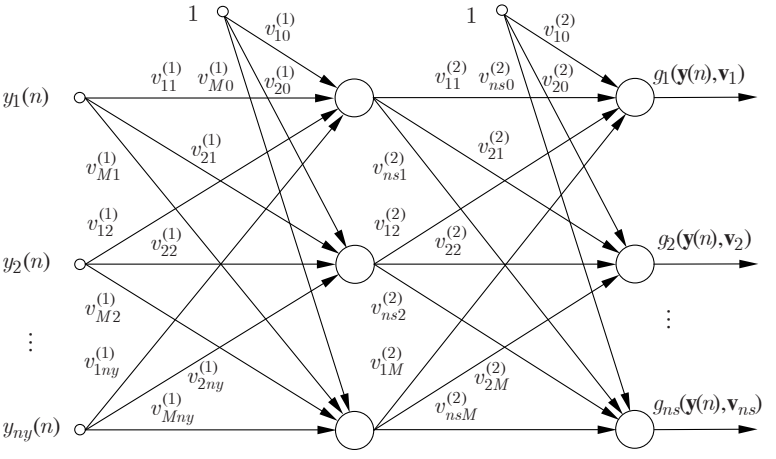

Fig. 2.7. Inverse neural network model of the MIMO nonlinear element

$$\hat{\mathbf{B}}^{(m)} \in \mathbb{R}^{ns \times nu}. \tag{2.28}$$

The series-parallel MIMO Wiener model is more complex than the parallel one as it uses an inverse model of the nonlinear element – Fig. 2.7. To derive the series-parallel MIMO Wiener model, assume that the nonlinear function $\mathbf{f}(\cdot)$ describing the nonlinear element is invertible. Then the output of the MIMO linear dynamic model can be expressed as

$$\hat{\mathbf{s}}(n) = -\sum_{m=1}^{na} \hat{\mathbf{A}}^{(m)} \hat{\mathbf{g}}\big(\mathbf{y}(n-m), \mathbf{V}\big) + \sum_{m=1}^{nb} \hat{\mathbf{B}}^{(m)} \mathbf{u}(n-m), \tag{2.29}$$

where

$$\hat{\mathbf{g}}\big(\mathbf{y}(n), \mathbf{V}\big) = \big[\hat{g}_1\big(\mathbf{y}(n), \mathbf{v}_1\big) \dots \hat{g}_{ns}\big(\mathbf{y}(n), \mathbf{v}_{ns}\big)\big]^T, \tag{2.30}$$

$$\mathbf{V} = \big[v_{10}^{(1)} \dots v_{Mny}^{(1)} \; v_{10}^{(2)} \dots v_{nsM}^{(2)}\big]^T, \tag{2.31}$$

$$\mathbf{v}_t = \big[v_{10}^{(1)} \dots v_{Mny}^{(1)} \; v_{t0}^{(2)} \dots v_{tM}^{(2)}\big]^T, \tag{2.32}$$

where $t = 1, \dots, ns$.

Note that $\hat{\mathbf{g}}(\cdot)$ in (2.29) describes the inverse nonlinear element model and does not denote the inverse of $\hat{\mathbf{f}}(\cdot)$. Assuming that the inverse nonlinear element is modelled by a single hidden layer perceptron with ny inputs, ns outputs, and M nonlinear nodes (Fig. 2.7), its tth output is

$$\hat{g}_t\big(\mathbf{y}(n), \mathbf{v}_t\big) = \sum_{j=1}^{M} v_{tj}^{(2)} \varphi\big(z_j(n)\big) + v_{t0}^{(2)}, \tag{2.33}$$

$$z_j(n) = \sum_{i=1}^{ny} v_{ji}^{(1)} y_i(n) + v_{j0}^{(1)}. \tag{2.34}$$

2.4 Gradient calculation

The calculation of the gradient in series-parallel Wiener models can be carried out with the well-known backpropagation algorithm. Series-parallel Wiener models are specialized multilayer perceptron networks with a multilayer perceptron model of the inverse nonlinear element followed by a single linear node model of the linear dynamic system followed by another multilayer perceptron structure of the nonlinear element model. In contrast, parallel Wiener models are dynamic systems and the corresponding neural networks are recurrent ones with one linear recurrent node that models the linear dynamic system followed by the multilayer perceptron model of the nonlinear element. This complicates the calculation of the gradient considerably. A crude approximation of the gradient can be obtained with the backpropagation method, which does not take into account the dynamic nature of the model. To evaluate the gradient more precisely, two other methods, known as the sensitivity method and the backpropagation through time method are used [88].
All pattern algorithms considered here have their batch counterparts. Given the rules of gradient computation, the batch versions of the algorithms can be derived easily, taking into account the fact that parameter updating should be performed iteratively, in a cumulative manner, after the presentation of all training patterns.

2.4.1 Series-parallel SISO model. Backpropagation method

Taking into account (2.9)–(2.13), the differentiation of (2.8) w.r.t. the model parameters $\hat{a}_k$, $k=1,\ldots,na$, $\hat{b}_k$, $k=1,\ldots,nb$, and v_c, w_c, $c=1,\ldots,3M+1$, where v_c and w_c denote the cth elements of $\mathbf{v}$ and $\mathbf{w}$, yields

$$\frac{\partial\hat{y}(n)}{\partial\hat{a}_k}=\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}\frac{\partial\hat{s}(n)}{\partial\hat{a}_k}=-\hat{g}\big(y(n-k),\mathbf{v}\big)\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}, \tag{2.35}$$

$$\frac{\partial\hat{y}(n)}{\partial\hat{b}_k}=\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}\frac{\partial\hat{s}(n)}{\partial\hat{b}_k}=u(n-k)\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}, \tag{2.36}$$

$$\frac{\partial\hat{y}(n)}{\partial v_c}=\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}\frac{\partial\hat{s}(n)}{\partial v_c}=-\sum_{m=1}^{na}\hat{a}_m\frac{\partial\hat{g}\big(y(n-m),\mathbf{v}\big)}{\partial v_c}\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}, \tag{2.37}$$

$$\frac{\partial\hat{y}(n)}{\partial w_c}=\frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial w_c}. \tag{2.38}$$

From (2.10) and (2.11), it follows that the partial derivative of the Wiener model output w.r.t. the output of the linear dynamic model is

$$\begin{aligned}\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)} &= \frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial\hat{s}(n)} = \sum_{j=1}^{M} w_{1j}^{(2)} \frac{\partial\varphi\big(x_j(n)\big)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial\hat{s}(n)} \\ &= \sum_{j=1}^{M} w_{j1}^{(1)} w_{1j}^{(2)} \varphi'\big(x_j(n)\big).\end{aligned} \tag{2.39}$$

Differentiating (2.12), partial derivatives of the inverse nonlinear element model output w.r.t. its parameters $v_{j1}^{(1)}$, $v_{j0}^{(1)}$, $v_{1j}^{(2)}$, $j = 1, \ldots, M$, and $v_{10}^{(2)}$ can be calculated as

$$\frac{\partial\hat{g}\big(y(n),\mathbf{v}\big)}{\partial v_{j1}^{(1)}} = \frac{\partial\hat{g}\big(y(n),\mathbf{v}\big)}{\partial\varphi\big(z_j(n)\big)} \frac{\partial\varphi\big(z_j(n)\big)}{\partial z_j(n)} \frac{\partial z_j(n)}{\partial v_{j1}^{(1)}} = v_{1j}^{(2)} \varphi'\big(z_j(n)\big) y(n), \tag{2.40}$$

$$\frac{\partial\hat{g}\big(y(n),\mathbf{v}\big)}{\partial v_{j0}^{(1)}} = \frac{\partial\hat{g}\big(y(n),\mathbf{v}\big)}{\partial\varphi\big(z_j(n)\big)} \frac{\partial\varphi\big(z_j(n)\big)}{\partial z_j(n)} \frac{\partial z_j(n)}{\partial v_{j0}^{(1)}} = v_{1j}^{(2)} \varphi'\big(z_j(n)\big), \tag{2.41}$$

$$\frac{\partial\hat{g}\big(y(n),\mathbf{v}\big)}{\partial v_{1j}^{(2)}} = \varphi\big(z_j(n)\big), \tag{2.42}$$

$$\frac{\partial\hat{g}\big(y(n),\mathbf{v}\big)}{\partial v_{10}^{(2)}} = 1. \tag{2.43}$$

From (2.10) and (2.11), it follows that partial derivatives of the output of the nonlinear element model w.r.t. the weights $w_{j1}^{(1)}$, $w_{j0}^{(1)}$, $w_{1j}^{(2)}$, $j = 1, \ldots, M$, and $w_{10}^{(2)}$ are

$$\frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial w_{j1}^{(1)}} = \frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial\varphi\big(x_j(n)\big)} \frac{\partial\varphi\big(x_j(n)\big)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial w_{j1}^{(1)}} = w_{1j}^{(2)} \varphi'\big(x_j(n)\big)\hat{s}(n), \tag{2.44}$$

$$\frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial w_{j0}^{(1)}} = \frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial\varphi\big(x_j(n)\big)} \frac{\partial\varphi\big(x_j(n)\big)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial w_{j0}^{(1)}} = w_{1j}^{(2)} \varphi'\big(x_j(n)\big), \tag{2.45}$$

$$\frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial w_{1j}^{(2)}} = \varphi\big(x_j(n)\big), \tag{2.46}$$

$$\frac{\partial\hat{f}\big(\hat{s}(n),\mathbf{w}\big)}{\partial w_{10}^{(2)}} = 1. \tag{2.47}$$

In spite of the fact that the series-parallel model is of the feedforward type, its training with the backpropagation (BPS) method is quite complex. This comes from the fact that the total number of hidden layers in the series-parallel model equals 4. Moreover, although both the nonlinear element and its inverse are identified, only an approximate inverse relationship between them is obtained in practice.

2.4.2 Parallel SISO model. Backpropagation method

The parallel Wiener model does not contain any inverse model of the nonlinear element (Fig. 2.5). This simplifies the training of the model considerably. On the other hand, as only an approximate gradient is calculated with the backpropagation (BPP) method, a very slow convergence rate can be observed. In the BPP method, the dependence of the past linear dynamic model outputs $\hat{s}(n-m), m = 1, \ldots, na$, on the parameters $\hat{a}_k$ and $\hat{b}_k$ is neglected. Hence, from (2.14) and (2.15), it follows that

$$\frac{\partial \hat{y}(n)}{\partial \hat{a}_k} = \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)} \frac{\partial \hat{s}(n)}{\partial \hat{a}_k} = -\hat{s}(n-k) \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)}, \ k = 1, \ldots, na, \tag{2.48}$$

$$\frac{\partial \hat{y}(n)}{\partial \hat{b}_k} = \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)} \frac{\partial \hat{s}(n)}{\partial \hat{b}_k} = u(n-k) \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)}, \ k = 1, \ldots, nb, \tag{2.49}$$

$$\frac{\partial \hat{y}(n)}{\partial w_c} = \frac{\partial \hat{f}\big(\hat{s}(n), \mathbf{w}\big)}{\partial w_c}. \tag{2.50}$$

The partial derivative of the parallel Wiener model output w.r.t. the output of the linear dynamic model is

$$\begin{aligned} \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)} &= \frac{\partial \hat{f}\big(\hat{s}(n), \mathbf{w}\big)}{\partial \hat{s}(n)} = \sum_{j=1}^{M} w_{1j}^{(2)} \frac{\partial \varphi\big(x_j(n)\big)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial \hat{s}(n)} \\ &= \sum_{j=1}^{M} w_{j1}^{(1)} w_{1j}^{(2)} \varphi'\big(x_j(n)\big). \end{aligned} \tag{2.51}$$

Partial derivatives of the parallel Wiener model output w.r.t. the weights $w_{j1}^{(1)}$, $w_{j0}^{(1)}$, $w_{1j}^{(2)}$, $j = 1, \ldots, M$, and $w_{10}^{(2)}$ are calculated in the same way as for the series-parallel model using (2.44) – (2.47).

2.4.3 Parallel SISO model. Sensitivity method

The sensitivity method (SM) differs from the BPP method markedly in the calculation of partial derivatives of the output of the linear dynamic model

w.r.t. its parameters. In general, as the SM uses a more accurate evaluation of the gradient than the BPP method, a higher convergence rate can be expected. Assuming that the parameters $\hat{a}_k$ and $\hat{b}_k$ do not change and differentiating (2.15), we have

$$\frac{\partial \hat{s}(n)}{\partial \hat{a}_k} = -\hat{s}(n-k) - \sum_{m=1}^{na} \hat{a}_m \frac{\partial \hat{s}(n-m)}{\partial \hat{a}_k}, \ k = 1, \ldots, na, \tag{2.52}$$

$$\frac{\partial \hat{s}(n)}{\partial \hat{b}_k} = u(n-k) - \sum_{m=1}^{na} \hat{a}_m \frac{\partial \hat{s}(n-m)}{\partial \hat{b}_k}, \ k = 1, \ldots, nb. \tag{2.53}$$

The partial derivatives (2.52) and (2.53) can be computed on-line by simulation, usually with zero initial conditions. The substitution of (2.52) and (2.53) into (2.48) and (2.49) gives

$$\frac{\partial \hat{y}(n)}{\partial \hat{a}_k} = \left(-\hat{s}(n-k) - \sum_{m=1}^{na} \hat{a}_m \frac{\partial \hat{s}(n-m)}{\partial \hat{a}_k} \right) \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)}, \tag{2.54}$$

$$\frac{\partial \hat{y}(n)}{\partial \hat{b}_k} = \left(u(n-k) - \sum_{m=1}^{na} \hat{a}_m \frac{\partial \hat{s}(n-m)}{\partial \hat{b}_k} \right) \frac{\partial \hat{y}(n)}{\partial \hat{s}(n)}. \tag{2.55}$$

The calculation of partial derivatives for the parallel Wiener model is illustrated in Fig. 2.8. Other partial derivatives are calculated in the SM in the same way as in the BPP method. In comparison with the BPP method, the SM is only a little more computationally intensive. The increase in computational burden comes from the simulation of $na + nb$ sensitivity models, whereas dynamic models of a low order are used commonly.

Note that to obtain the exact value of the gradient, the parameters $\hat{a}_k$ and $\hat{b}_k$ should be kept constant. That is the case only in the batch mode, in which the parameters are updated after the presentation of all learning patterns. In the sequential mode (pattern learning), the parameters $\hat{a}_k$ and $\hat{b}_k$ are updated after each learning pattern and, as a result, an approximate value of the gradient is obtained. Therefore, to achieve a good approximation accuracy, the learning rate η should be sufficiently small to keep changes in the parameters negligible.

2.4.4 Parallel SISO model. Backpropagation through time method

In the backpropagation through time (BPTT) method, partial derivatives of the model output w.r.t. the weights of the nonlinear element model are calculated in the same way as in BPS and BPP methods, or the SM from (2.44) – (2.47). Also, partial derivatives of the model output w.r.t. the output of the linear dynamic model are calculated in the same way from (2.51).

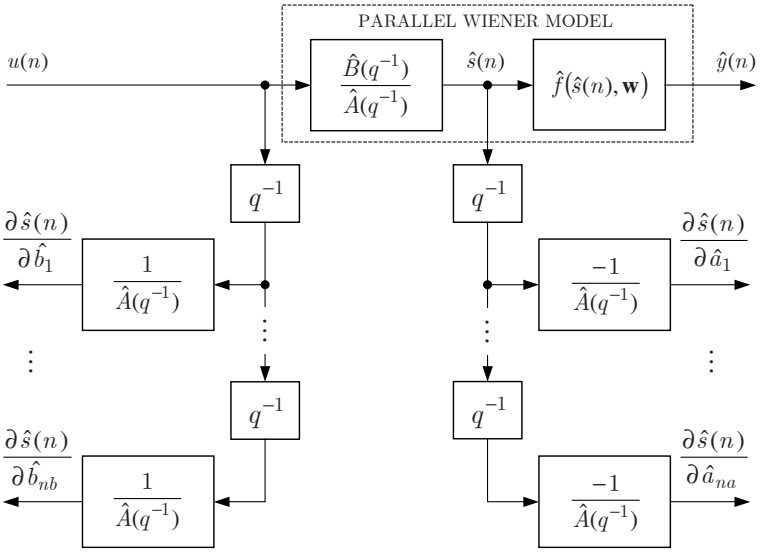

Fig. 2.8. Parallel Wiener model and its sensitivity models

The only difference is in the computation of partial derivatives of the linear dynamic model output w.r.t. its parameters $\hat{a}_k$ and $\hat{b}_k$. To perform this, the linear dynamic model is unfolded back in time. Unfolding (2.15) back in time for one step gives

$$\begin{aligned}\hat{s}(n) = &- \hat{a}_1\hat{s}(n-1) - \sum_{m=2}^{na}\hat{a}_m\hat{s}(n-m) + \sum_{m=1}^{nb}\hat{b}_m u(n-m) = \\ &- \hat{a}_1\Big[-\sum_{m=1}^{na}\hat{a}_m\hat{s}(n-m-1) + \sum_{m=1}^{nb}\hat{b}_m u(n-m-1)\Big] \\ &- \sum_{m=2}^{na}\hat{a}_m\hat{s}(n-m) + \sum_{m=1}^{nb}\hat{b}_m u(n-m).\end{aligned} \tag{2.56}$$

Such an unfolding procedure can be continued until the initial time step is obtained. The unfolded Wiener model is no more of the recurrent type and can be represented by a feedforward neural network with the model of the nonlinear element on its top and copies of the linear dynamic model below, see Fig. 2.9 for an example of the 3rd order model. When discrete time elapses, the number of copies of the linear dynamic model increases. At the time n, a fully unfolded model contains n copies of the linear dynamic model. To keep computational complexity constant, unfolding in time is commonly restricted to only a given number of time steps K in a method called truncated

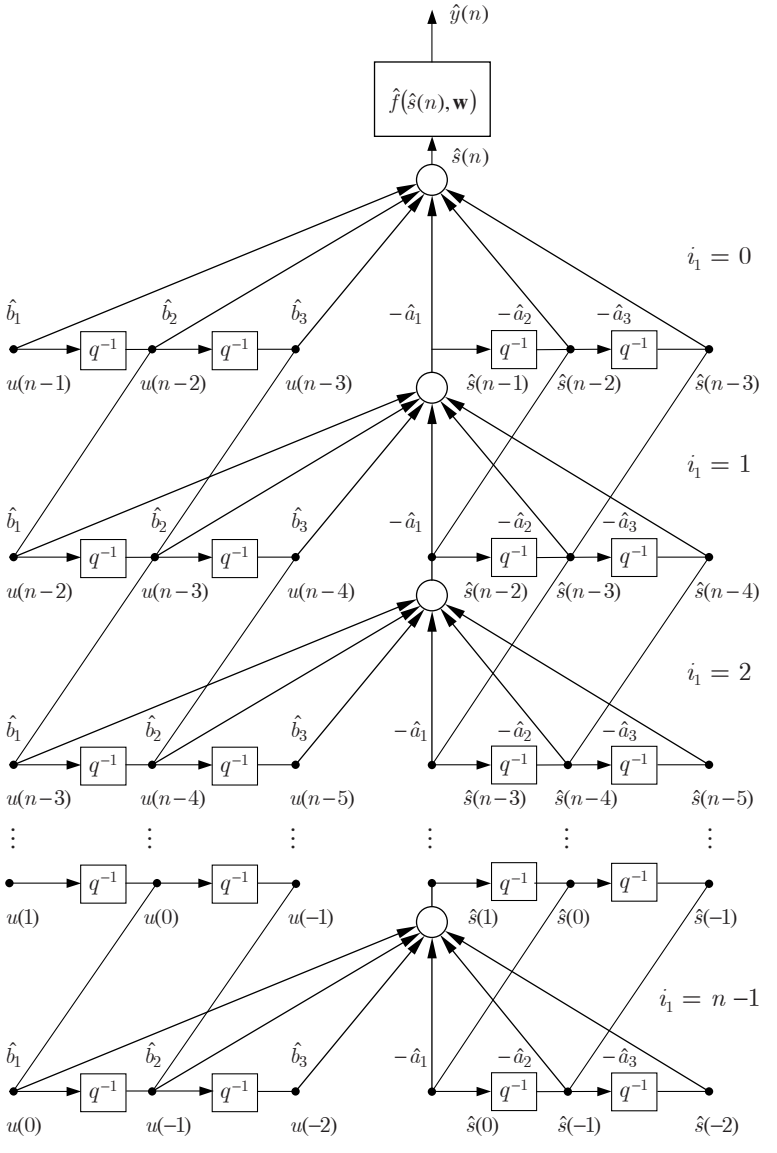

Fig. 2.9. Unfolded-in-time Wiener model of the third order

BPTT. In this way, only an approximate gradient is computed. The unfolded network is of the feedforward type and differs from a common multilayer perceptron distinctly. First, apart from adjustable weights, marked in Fig. 2.9 with thick lines, it contains nonadjustable short-circuit connections between the neighboring layers, marked with thin lines. Then for models of the order

$na > 2$, some of these short-circuit weights are connected in series and form lines connecting more distant layers. Finally, the unfolded model contains as many copies of the linear dynamic model as unfolded time steps. All these differences should be taken into account in the training algorithm. A detailed description of the truncated BPTT algorithm is given in Appendix 2.1.

2.4.5 Series-parallel MIMO model. Backpropagation method

In a sequential version of the gradient-based learning algorithm extended to the MIMO case, the following cost function is minimized w.r.t. the model parameters:

$$J(n) = \frac{1}{2}\sum_{t=1}^{ny}\big(y_t(n) - \hat{y}_t(n)\big)^2, \tag{2.57}$$

where $\hat{y}_t(n) = \hat{f}_t\big(\hat{\mathbf{s}}(n),\mathbf{w}_t\big)$ is the tth output of the Wiener model, and $\hat{\mathbf{s}}(n)$ denotes the vector of the outputs of the linear dynamic model. From (2.29), it follows that the lth output of the linear dynamic model can be written as

$$\hat{s}_l(n) = -\sum_{m=1}^{na}\sum_{m_1=1}^{ns}\hat{a}_{lm_1}^{(m)}\hat{g}_{m_1}\big(\mathbf{y}(n-m),\mathbf{v}_{m_1}\big) + \sum_{m=1}^{nb}\sum_{m_1=1}^{nu}\hat{b}_{lm_1}^{(m)}u_{m_1}(n-m), \tag{2.58}$$

where $l = 1,\ldots,ns$, and $\hat{a}_{lm_1}^{(m)}$ and $\hat{b}_{lm_1}^{(m)}$ denote the elements of $\hat{\mathbf{A}}^{(m)}$ and $\hat{\mathbf{B}}^{(m)}$. Partial derivatives of the series-parallel model (2.17) and (2.29) can be calculated with the backpropagation method (BPS). Differentiating (2.23), partial derivatives of the model output $\hat{y}_t(n)$, $t = 1,\ldots,ny$, w.r.t. the parameters of the nonlinear element model can be obtained as

$$\frac{\partial\hat{y}_t(n)}{\partial w_{ji}^{(1)}} = \frac{\partial\hat{f}_t\big(\hat{\mathbf{s}}(n),\mathbf{w}_t\big)}{\partial\varphi\big(x_j(n)\big)}\frac{\partial\varphi\big(x_j(n)\big)}{\partial x_j(n)}\frac{\partial x_j(n)}{\partial w_{ji}^{(1)}} = w_{tj}^{(2)}\varphi'\big(x_j(n)\big)\hat{s}_i(n), \tag{2.59}$$

$$\frac{\partial\hat{y}_t(n)}{\partial w_{j0}^{(1)}} = \frac{\partial\hat{f}_t\big(\hat{\mathbf{s}}(n),\mathbf{w}_t\big)}{\partial\varphi\big(x_j(n)\big)}\frac{\partial\varphi\big(x_j(n)\big)}{\partial x_j(n)}\frac{\partial x_j(n)}{\partial w_{j0}^{(1)}} = w_{tj}^{(2)}\varphi'\big(x_j(n)\big), \tag{2.60}$$

$$\frac{\partial\hat{y}_t(n)}{\partial w_{tj}^{(2)}} = \varphi\big(x_j(n)\big), \tag{2.61}$$

$$\frac{\partial\hat{y}_t(n)}{\partial w_{t0}^{(2)}} = 1, \tag{2.62}$$

where $j = 1,\ldots,M$, and $i = 1,\ldots,ns$. From (2.23) and (2.58), it follows that

$$\frac{\partial\hat{y}_t(n)}{\partial\hat{a}_{lm_1}^{(k)}} = \frac{\partial\hat{y}_t(n)}{\partial\hat{s}_l(n)}\frac{\partial\hat{s}_l(n)}{\partial\hat{a}_{lm_1}^{(k)}} = -\hat{g}_{m_1}\big(\mathbf{y}(n-k),\mathbf{v}_{m_1}\big)\frac{\partial\hat{y}_t(n)}{\partial\hat{s}_l(n)}, \quad m_1 = 1,\ldots,ns, \quad k = 1,\ldots,na, \tag{2.63}$$

$$\frac{\partial \hat{y}_t(n)}{\partial \hat{b}^{(k)}_{lm_1}} = \frac{\partial \hat{y}_t(n)}{\partial \hat{s}_l(n)} \frac{\partial \hat{s}_l(n)}{\partial \hat{b}^{(k)}_{lm_1}} = u_{m_1}(n-k) \frac{\partial \hat{y}_t(n)}{\partial \hat{s}_l(n)}, \quad m_1 = 1, \ldots, nu, \quad k = 1, \ldots, nb. \tag{2.64}$$

From (2.23) and (2.24), it follows that

$$\begin{aligned} \frac{\partial \hat{y}_t(n)}{\hat{s}_l(n)} &= \frac{\partial \hat{f}_t(\hat{\mathbf{s}}(n), \mathbf{w}_t)}{\partial \hat{s}_l(n)} = \sum_{j=1}^{M} w^{(2)}_{tj} \frac{\partial \varphi\big(x_j(n)\big)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial \hat{s}_l(n)} \\ &= \sum_{j=1}^{M} w^{(1)}_{jl} w^{(2)}_{tj} \varphi'\big(x_j(n)\big). \end{aligned} \tag{2.65}$$

Taking into account (2.58), partial derivatives of the model output w.r.t. the parameters of the inverse nonlinear element model can be calculated according to the following rule:

$$\begin{aligned} \frac{\partial \hat{y}_t(n)}{\partial v} &= \frac{\partial \hat{f}_t\big(\mathbf{s}(n), \mathbf{w}_t\big)}{\partial v} = \sum_{l=1}^{ns} \frac{\partial \hat{y}_t(n)}{\hat{s}_l(n)} \frac{\partial \hat{s}_l(n)}{\partial v} \\ &= \sum_{l=1}^{ns} \frac{\partial \hat{y}_t(n)}{\hat{s}_l(n)} \sum_{m=1}^{na} \sum_{m_1=1}^{ns} \hat{a}^{(m)}_{lm_1} \frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n-m), \mathbf{v}_{m_1}\big)}{\partial v}, \end{aligned} \tag{2.66}$$

where v is any parameter of the inverse nonlinear element model. From (2.33) and (2.34), it follows that partial derivatives of the inverse nonlinear element output w.r.t. its parameters are

$$\begin{aligned} \frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n), \mathbf{v}_{m_1}\big)}{\partial v^{(1)}_{ji}} &= \frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n), \mathbf{v}_{m_1}\big)}{\partial \varphi\big(z_j(n)\big)} \frac{\partial \varphi\big(z_j(n)\big)}{\partial z_j(n)} \frac{\partial z_j(n)}{\partial v^{(1)}_{ji}} \\ &= v^{(2)}_{m_1 j} \varphi'\big(z_j(n)\big) y_i(n), \end{aligned} \tag{2.67}$$

$$\begin{aligned} \frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n), \mathbf{v}_{m_1}\big)}{\partial v^{(1)}_{j0}} &= \frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n), \mathbf{v}_{m_1}\big)}{\partial \varphi\big(z_j(n)\big)} \frac{\partial \varphi\big(z_j(n)\big)}{\partial z_j(n)} \frac{\partial z_j(n)}{\partial v^{(1)}_{j0}} \\ &= v^{(2)}_{m_1 j} \varphi'\big(z_j(n)\big), \end{aligned} \tag{2.68}$$

$$\frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n), \mathbf{v}_{m_1}\big)}{\partial v^{(2)}_{m_1 j}} = \varphi\big(z_j(n)\big), \tag{2.69}$$

$$\frac{\partial \hat{g}_{m_1}\big(\mathbf{y}(n), \mathbf{v}_{m_1}\big)}{\partial v^{(2)}_{m_1 0}} = 1, \tag{2.70}$$

where $m_1 = 1, \ldots, ns$, $j = 1, \ldots, M$, $i = 1, \ldots, ny$.

2.4.6 Parallel MIMO model. Backpropagation method

From (2.25), it follows that the lth output of the linear dynamic model can be written as

$$\hat{s}_l(n) = -\sum_{m=1}^{na}\sum_{m_1=1}^{ns}\hat{a}_{lm_1}^{(m)}\hat{s}_{m1}(n-m) + \sum_{m=1}^{nb}\sum_{m_1=1}^{nu}\hat{b}_{lm_1}^{(m)}u_{m1}(n-m), \qquad (2.71)$$

where $l = 1,\ldots,ns$. Applying backpropagation rules to the parallel MIMO Wiener model, the dependence of the delayed outputs $\hat{s}_{m_1}(n-m)$, $m = 1,\ldots,na$, on the parameters $\hat{a}_{lm_1}^{(m)}$ and $\hat{b}_{lm1}^{(m)}$ is neglected. This reduces computational complexity of the backpropagation method for the parallel model (BPP) in comparison with the BPS method as no inverse nonlinear model is utilized. Using the BPP method, the calculation of partial derivatives of the model output w.r.t. the parameters of the nonlinear element model is performed in the same way as in the BPS method according to the formulae (2.59) – (2.62). Also, the calculation of $\partial\hat{y}_t(n)/\partial\hat{b}_{lm_1}^{(k)}$ is performed in the same way as in the BPS method according to (2.64). As the parallel Wiener model uses delayed outputs of the linear dynamic model instead of the outputs of the inverse nonlinear model, BPP differs from BPS in the calculation of $\partial\hat{y}_t(n)/\partial\hat{a}_{lm_1}^{(k)}$:

$$\frac{\partial\hat{y}_t(n)}{\partial\hat{a}_{lm_1}^{(k)}} = \frac{\partial\hat{y}_t(n)}{\partial\hat{s}_l(n)}\frac{\partial\hat{s}_l(n)}{\partial\hat{a}_{lm_1}^{(k)}} = -\hat{s}_{m_1}(n-k)\frac{\partial\hat{y}_t(n)}{\partial\hat{s}_l(n)},\ k = 1,\ldots,na. \qquad (2.72)$$

2.4.7 Parallel MIMO model. Sensitivity method

Assume that the linear dynamic model is time invariant. As in the case of the BPS and BPP methods for MIMO Wiener models, the calculation of partial derivatives of model output w.r.t. the parameters of the nonlinear element model in the SM is performed in the same way. From (2.71), it follows that to calculate $\partial\hat{y}_t(n)/\partial\hat{a}_{rp}^{(k)}$, $t = 1,\ldots,ny$, $k = 1,\ldots,na$, $r = 1,\ldots,ns$, $p = 1,\ldots,ns$, and $\partial\hat{y}_t(n)/\partial\hat{b}_{rp}^{(k)}$, $t = 1,\ldots,ny$, $k = 1,\ldots,nb$, $r = 1,\ldots,ns$, $p = 1,\ldots,nu$, the following set of linear difference equations is solved on-line by simulation:

$$\frac{\partial\hat{s}_l(n)}{\partial\hat{a}_{rp}^{(k)}} = -\delta_{lr}\hat{s}_p(n-k) - \sum_{m=1}^{na}\sum_{m_1=1}^{ns}\hat{a}_{lm_1}^{(m)}\frac{\partial\hat{s}_{m_1}(n-m)}{\partial\hat{a}_{rp}^{(k)}}, \qquad (2.73)$$

$$\frac{\partial\hat{s}_l(n)}{\partial\hat{b}_{rp}^{(k)}} = \delta_{lr}u_p(n-k) - \sum_{m=1}^{na}\sum_{m_1=1}^{ns}\hat{a}_{lm_1}^{(m)}\frac{\partial\hat{s}_{m_1}(n-m)}{\partial\hat{b}_{rp}^{(k)}}, \qquad (2.74)$$

where

$$\delta_{lr} = \begin{cases} 1 & \text{for } l = r \\ 0 & \text{for } l \neq r \end{cases}. \tag{2.75}$$

To solve (2.73) and (2.74), zero initial conditions are assumed.

2.4.8 Parallel MIMO model. Backpropagation through time method

Detailed derivation of the truncated backpropagation through time (BPTT) method for the parallel MIMO Wiener model is given in Appendix 2.2.

2.4.9 Accuracy of gradient calculation with truncated BPTT

An important issue that arises in the practical application of truncated BPTT is a proper choice of the number of unfolded time steps K. The number K should be large enough to ensure a good approximation of the gradient. On the other hand, too large K does not improve the convergence rate significantly but it increases computational complexity of the algorithm. Theorems 2.1 and 2.2 below give some insight into gradient calculation accuracy on the basis of the linear dynamic model and its sensitivity models and allows one to draw a conclusion on how the choice of K is to be made [89].

Theorem 2.1. Define the computation error

$$\Delta\hat{s}_{\hat{a}_k}(n) = \frac{\partial\hat{s}(n)}{\partial\hat{a}_k} - \frac{\partial^+\hat{s}(n)}{\partial\hat{a}_k},$$

where $\partial\hat{s}(n)/\partial\hat{a}_k$, $k = 1, \ldots, na$, denote partial derivatives calculated with the BPTT method, i.e. unfolding the model (2.15) $n-1$ times back in time, and $\partial^+\hat{s}(n)/\partial\hat{a}_k$ denote partial derivatives calculated with the truncated BPTT method unfolding the model (2.15) K times back in time.

Assume that

(A1) The linear dynamic model $\hat{B}(q^{-1})/\hat{A}(q^{-1})$ is asymptotically stable;
(A2) $\hat{s}(n) = 0$, $n = 0, \ldots, -na+1$; $u(n) = 0$, $n = -1, \ldots, -nb$;
(A3) $\partial\hat{s}(n)/\partial\hat{a}_k = 0$, $n = 0, \ldots, -na+1$;
(A4) The input $u(n)$, $n = 0, 1, \ldots$, is a sequence of zero-mean i.i.d. random variables of finite moments,

$$\mathrm{E}[u(n)] = 0,$$
$$\mathrm{E}[u(n)u(m)] = \begin{cases} \sigma^2 & \text{for } n = m \\ 0 & \text{for } n \neq m \end{cases}.$$

Then

$$\mathrm{var}\big(\Delta\hat{s}_{\hat{a}_k}(n)\big) = \sigma^2 \sum_{i_1=K+1}^{n-k} \left(\sum_{i_2=K+1}^{i_1} h_1(i_2)h_2(i_1-i_2) \right)^2, \; k = 1, \ldots, na, \tag{2.76}$$

for $n > K + k$ and 0 otherwise, where $h_1(n)$ is the impulse response function of the system

$$H_1(q^{-1}) = \frac{1}{\hat{A}(q^{-1})}, \tag{2.77}$$

and $h_2(n)$ is the impulse response function of the system

$$H_2(q^{-1}) = \frac{\hat{B}(q^{-1})}{\hat{A}(q^{-1})}. \tag{2.78}$$

Proof: The proof is shown in Appendix 2.3.

Theorem 2.2. Define the computation error

$$\Delta\hat{s}_{\hat{b}_k}(n) = \frac{\partial\hat{s}(n)}{\partial\hat{b}_k} - \frac{\partial^+\hat{s}(n)}{\partial\hat{b}_k},$$

where $\partial\hat{s}(n)/\partial\hat{b}_k$, $k = 1, \ldots, nb$, denote partial derivatives calculated with the BPTT method, i.e. unfolding the model (2.15) $n-1$ times back in time, and $\partial^+\hat{s}(n)/\partial\hat{b}_k$ denote partial derivatives calculated with the truncated BPTT method unfolding the model (2.15) K times back in time.

Assume that

(A1) The linear dynamic model $\hat{B}(q^{-1})/\hat{A}(q^{-1})$ is asymptotically stable;
(A2) $\hat{s}(n) = 0$, $n = 0, \ldots, -na + 1$; $u(n) = 0$, $n = -1, \ldots, -nb$;
(A3) $\partial\hat{s}(n)/\partial\hat{b}_k = 0$, $n = 0, \ldots, -na + 1$;
(A4) The input $u(n)$, $n = 0, 1, \ldots$, is a sequence of zero-mean i.i.d. random variables of finite moments,

$$\mathrm{E}[u(n)] = 0,$$
$$\mathrm{E}[u(n)u(m)] = \begin{cases} \sigma^2 & \text{for } n = m \\ 0 & \text{for } n \neq m \end{cases}.$$

Then

$$\mathrm{var}\big(\Delta\hat{s}_{\hat{b}_k}(n)\big) = \sigma^2 \sum_{i_1=K+1}^{n-k} h_1^2(i_1),\ k = 1, \ldots, nb, \tag{2.79}$$

for $n > K + k$ and 0 otherwise, where $h_1(n)$ is the impulse response function of the system

$$H_1(q^{-1}) = \frac{1}{\hat{A}(q^{-1})}. \tag{2.80}$$

Remark 2.1. From (2.76) and (2.79), it follows that the variances of computation errors do not depend on k for $n \to \infty$,

$$\lim_{n\to\infty} \operatorname{var}\big(\Delta\hat{s}_{\hat{a}_k}(n)\big) = \sigma^2 \sum_{i_1=K+1}^{\infty} \left(\sum_{i_2=K+1}^{i_1} h_1(i_2)h_2(i_1-i_2) \right)^2, \tag{2.81}$$

$$\lim_{n\to\infty} \operatorname{var}\big(\Delta\hat{s}_{\hat{b}_k}(n)\big) = \sigma^2 \sum_{i_1=K+1}^{\infty} h_1^2(i_1). \tag{2.82}$$

Theorems 2.1 and 2.2 provide a useful tool for the determination of the number of unfolded time steps K. From Theorems 2.1 and 2.2, it follows that, for a fixed value of K, the accuracy of gradient calculation with the truncated BPTT method depends on the number of discrete time steps necessary for the impulse response $h_1(n)$ to decrease to negligible small values. Note that this impulse response can differ significantly from an impulse response of the system $1/A(q^{-1})$, particularly at the beginning of the learning process. Therefore, both $h_1(n)$ and $h_2(n)$ can be traced during the training of the neural network model and K can be changed adaptively to meet the assumed degrees of the gradient calculation accuracy $\xi_{a_k}(n)$ and $\xi_{b_k}(n)$ defined as the ratio of the variances (2.76) or (2.79) to the variances of the corresponding partial derivatives:

$$\xi_{a_k}(n) = \frac{\operatorname{var}\big(\Delta\hat{s}_{\hat{a}_k}(n)\big)}{\operatorname{var}\left(\dfrac{\partial\hat{s}(n)}{\partial\hat{a}_k}\right)} = \frac{\displaystyle\sum_{i_1=K+1}^{n-k} \left(\sum_{i_2=K+1}^{i_1} h_1(i_2)h_2(i_1-i_2) \right)^2}{\displaystyle\sum_{i_1=0}^{n-k} \left(\sum_{i_2=0}^{i_1} h_1(i_2)h_2(i_1-i_2) \right)^2}, \tag{2.83}$$

$$\xi_{b_k}(n) = \frac{\operatorname{var}\big(\Delta\hat{s}_{\hat{b}_k}(n)\big)}{\operatorname{var}\left(\dfrac{\partial\hat{s}(n)}{\partial\hat{b}_k}\right)} = \frac{\displaystyle\sum_{i_1=K+1}^{n-k} h_1^2(i_1)}{\displaystyle\sum_{i_1=0}^{n-k} h_1^2(i_1)}. \tag{2.84}$$

The initial value of K can be determined based on the assumed initial values of $\hat{a}_k$ and $\hat{b}_k$ or a priori knowledge of system dynamics.
The required value of K depends strongly on the system sampling rate. If the sampling rate increases for a given system, the number of discrete time steps, necessary for the impulse response $h_1(n)$ to decrease to negligible small values, increases and a higher value of K is necessary to achieve the same accuracy of gradient calculation.

2.4.10 Gradient calculation in the sequential mode

The derivation of gradient calculation algorithms has been made under the assumption that a model is time invariant. Obviously, pattern by pattern

updating of model parameters in the sequential mode makes this assumption no longer true. As a result, approximated values of the gradient are obtained for parallel models. To obtain a good approximation of the gradient, the learning rate should be small enough to achieve negligible small changes of model parameters. In comparison with the SM, an important advantage of the BPTT method is that the actual values of model parameters can be used in the unfolding procedure. Although the gradient calculated in this way is still approximate, such an approach may increase the convergence rate of the learning process, as illustrated in the simulation example below, even if model unfolding is restricted to only a few time steps. Another interesting feature of the BPTT method is that it can provide exact values of the gradient when not only the actual values of parameters are employed but also linear dynamic model outputs are unfolded back in time according to the following rule:

$$\hat{s}(n-na-i_1) = \frac{1}{\hat{a}_{na}}\Bigg(- \hat{s}(n-i_1) - \sum_{m=1}^{na-1} \hat{a}_m \hat{s}(n-m-i_1) + \sum_{m=1}^{nb} \hat{b}_m u(n-m-i_1)\Bigg), \tag{2.85}$$

where $i_1 = 1, \ldots, n-1$. The formula (2.85) can be used provided that $\hat{a}_{na} \neq 0$. In practice, if $\hat{a}_{na} = 0$ or $\hat{a}_{na} \simeq 0$, then $\hat{s}(n-na-i_1)$ does not influence or does not influence significantly $\hat{s}(n-i_1)$, and the unfolding step can be omitted. Note that, provided that $\hat{b}_{nb} \neq 0$, unfolding linear dynamic model outputs can be replaced with unfolding model inputs:

$$u(n-nb-i_1) = \frac{1}{\hat{b}_{nb}}\Bigg(\hat{s}(n-i_1) + \sum_{m=1}^{na} \hat{a}_m \hat{s}(n-m-i_1) - \sum_{m=1}^{nb-1} \hat{b}_m u(n-m-i_1)\Bigg). \tag{2.86}$$

In this case, if $\hat{b}_{nb} = 0$ or $\hat{b}_{nb} \simeq 0$, then $u(n-na-i_1)$ does not influence or does not influence significantly $\hat{s}(n-i_1)$, and the unfolding step can be omitted.

2.4.11 Computational complexity

Computational complexity of on-line learning algorithms for recurrent neural network models is commonly much higher than computational complexity of the on-line backpropagation algorithm and depends on the number of the recurrent nodes L. For example, for the fully connected recurrent network, computational complexity per one step of BPTT, truncated up to K time steps backwards, is $O(L^2K)$. Computational complexity of the real time recurrent learning algorithm of Williams and Zipser [167] is even much higher $O(L^4)$ [115]. Fortunately, the parallel neural network Wiener model contains

only one recurrent node, i.e., $L = 1$. This considerably reduces computational requirements of the examined learning algorithms. The evaluation of computational complexity per one step given below is expressed in terms of the polynomial orders na and nb, and the number of the nonlinear nodes M. For the truncated BBTT algorithm, computational complexity depends also on the number of unfolded time steps K. This evaluation of computational complexity is made under the following assumptions:

1. The number of bits of precision for all quantities is fixed.
2. The costs of storing and reading from computer memory can be neglected.
3. No distinction is made between the costs of different operations such as addition, multiplication, calculation of input-output transfer functions such as $\varphi(\cdot)$ or its first derivative.

The computation of the model output requires the function $\varphi(\cdot)$ to be calculated M times, and to compute the gradient of the model output, its first derivative has to be calculated M times. Both of these operations are included into computational costs of the weight update given in Table 2.1. Computational complexity of BPS is approximately two times higher than computational complexity of BPP. The application of the SM requires only $2na(na + nb)$ operations more than BPP. Computational complexity of the BPTT algorithm depends also on the number of unfolded time steps K and is higher than the complexity of the BPP algorithm by $2K(2na + nb)$ operations.

Table 2.1. Computational complexity of on-line learning algorithms

Algorithm	Computational complexity
BPS	$(31 + 2na)M + 5(na + nb) + 5$
BPP	$16M + 6(na + nb) + 3$
SM	$16M + 2na(na + nb) + 6(na + nb) + 3$
BPTT	$16M + 2K(2na + nb) + 6(na + nb) + 3$

2.5 Simulation example

In the simulation example, the second order Wiener system composed of a continuous linear dynamic system given by the pulse transfer function

$$G(s) = \frac{1}{6s^2 + 5s + 1} \tag{2.87}$$

was converted to discrete time, assuming a zero order hold on the input and the sampling interval 1s, leading to the following difference equation:

$$s(n) = 1.3231s(n–1) – 0.4346s(n – 2) + 0.0635u(n–1) + 0.0481u(n – 2). \tag{2.88}$$

The system contained also a nonlinear element (Fig. 2.10) given by

$$f\big(s(n)\big) = \begin{cases} -1 & \text{for } s(n) < -1.5 \\ s(n) + 0.5 & \text{for } -1.5 \leqslant s(n) \leqslant -0.5 \\ 0 & \text{for } \mid s(n) \mid < 0.5 \\ s(n) - 0.5 & \text{for } 0.5 \leqslant s(n) \leqslant 1.5 \\ 1 & \text{for } s(n) > 1.5 \end{cases} . \tag{2.89}$$

The system was driven by a sequence of 60000 random numbers uniformly distributed in $(-2, 2)$. The parallel neural network Wiener model was trained recursively using the steepest descent method, with the learning rate of 0.005, and calculating the gradient with the BPP, truncated BPTT, and SM algorithms. The nonlinear element model contained 25 nodes of the hyperbolic tangent activation. To compare convergence rates of the algorithms, a mean square error of moving averages defined as

$$J_I(n) = \begin{cases} \dfrac{1}{n} \displaystyle\sum_{j=1}^{n} \big(\hat{y}(j) - y(j)\big)^2 & \text{for } n \leqslant I \\ \dfrac{1}{I} \displaystyle\sum_{j=n-I+1}^{n} \big(\hat{y}(j) - y(j)\big)^2 & \text{for } n > I \end{cases} \tag{2.90}$$

is used. The indices $P(n)$ and $F(n)$ are used to compare the identification accuracy of the linear dynamic system and the nonlinear function $f(\cdot)$, respectively,

$$P(n) = \frac{1}{4} \sum_{j=1}^{2} \big[(\hat{a}_j - a_j)^2 + (\hat{b}_j - b_j)^2\big], \tag{2.91}$$

$$F(n) = \frac{1}{100} \sum_{j=1}^{100} \big[\hat{f}(s(j)) - f(s(j), \mathbf{w})\big]^2, \tag{2.92}$$

where $\{s(j)\}$ is a testing sequence consisting of 100 linearly equally spaced values between -2 and 2. The simulation results for a noise-free Wiener system and a Wiener system disturbed by the additive output Gaussian noise $\mathcal{N}(0, \sigma_e)$ are summarized in Tables 2.2 – 2.4.

Noise-free case. The identification results are given in Table 2.2 and illustrated in Figs. 2.10 – 2.16. As shown in Table 2.2, the highest accuracy of nonlinear element identification, measured by the $F(60000)$ index, was achieved for the BPTT algorithm at $K = 6$. The best fitting of the linear dynamic model, expressed by the $P(60000)$ index, can be observed for the the BPTT algorithm, at $K = 4$. The moving averages index $J_{2000}(60000)$ that describes the accuracy of overall model identification also has its minimum for the

BPTT algorithm, at $K = 4$. The lowest accuracy of identification, measured by all four indices, was obtained for the BPP algorithm. The results obtained for the SM algorithm are more accurate in comparison with the results of the BPTT algorithm at $K = 1$ and less accurate than these at $K = 2, \ldots, 8$.

Noise case I. The results obtained at a high signal to noise ratio $SNR = 17.8$, defined as $SNR = \sqrt{\text{var}(y(n) - \varepsilon(n))/\text{var}(\varepsilon(n))}$, are a little less accurate than these for the noise-free case (Table 2.3). The minimum values of all four indices were obtained for the BPTT algorithm, i.e., $F(60000)$ at $K = 6$, $P(60000)$ at $K = 5$, $J_{2000}(60000)$ at $K = 4$, and $J_{60000}(60000)$ at $K = 3$ and 4.

Noise case II. The results are given in Table 2.4. In this case, training the neural network Wiener model at a low signal to noise ratio $SNR = 3.56$ was made using the following time-dependent learning rate:

$$\eta(n) = \frac{0.01}{\sqrt[4]{n}}. \tag{2.93}$$

In general, the BPP algorithm has the lowest convergence rate. This result can be explained by employing in BPP the approximate gradient instead of the true one. In the batch mode, the truncated BPTT algorithm is an approximation of the SM algorithm and the gradient approximation error decreases with an increase in K and approaches 0 for the fully unfolded model. In the sequential mode, the actual values of linear dynamic model parameters are used to unfold the model back in time. Therefore, as the truncated BPTT algorithm is not a simple approximation of the SM algorithm, the BPTT estimation results do not converge to the results obtained for the SM algorithm. Note that, unlike the SM algorithm, BPTT makes it possible to calculate the true value of the gradient if the linear dynamic model output $s(n)$ is unfolded back in time.

Pattern learning is usually more convenient and effective when the number of training patterns N is large, which is the case in the simulation example.

Another advantage of pattern learning is that it explores the parameter space in a stochastic way by its nature, thus preventing the learning process from getting trapped in a shallow local minimum [68]. In practice, the number of the available training patterns may be too small to achieve the minimum of the global cost function J after a single presentation of all N training patterns. In such a situation, the learning process can be continued with the same set of N training patterns presented cyclically. Each cycle of repeated pattern learning corresponds to one iteration (epoch) in the batch mode.

Table 2.2. Comparison of estimation accuracy, ($\sigma_e = 0$)

Algorithm	$J_{60000}(60000)$	$J_{2000}(60000)$	$F(60000)$	$P(60000)$
BPP	5.51×10^{-3}	1.29×10^{-3}	8.01×10^{-3}	2.07×10^{-3}
SM	1.40×10^{-3}	7.94×10^{-5}	3.53×10^{-4}	2.66×10^{-6}
BPTT, $K = 1$	2.17×10^{-3}	3.80×10^{-4}	8.05×10^{-4}	4.16×10^{-5}
BPTT, $K = 2$	1.34×10^{-3}	7.52×10^{-5}	2.37×10^{-4}	4.12×10^{-7}
BPTT, $K = 3$	1.19×10^{-3}	3.81×10^{-5}	1.15×10^{-4}	6.81×10^{-7}
BPTT, $K = 4$	1.19×10^{-3}	2.71×10^{-5}	4.68×10^{-5}	3.82×10^{-7}
BPTT, $K = 5$	1.22×10^{-3}	2.78×10^{-5}	4.66×10^{-5}	9.66×10^{-7}
BPTT, $K = 6$	1.25×10^{-3}	2.86×10^{-5}	4.37×10^{-5}	1.74×10^{-6}
BPTT, $K = 7$	1.27×10^{-3}	2.97×10^{-5}	4.54×10^{-5}	1.35×10^{-6}
BPTT, $K = 8$	1.31×10^{-3}	3.99×10^{-5}	9.23×10^{-5}	2.00×10^{-6}
BPTT, $K = 9$	1.35×10^{-3}	5.25×10^{-5}	9.12×10^{-5}	4.10×10^{-6}
BPTT, $K = 10$	1.38×10^{-3}	9.62×10^{-5}	3.50×10^{-4}	3.05×10^{-6}
BPTT, $K = 11$	1.38×10^{-3}	9.03×10^{-5}	4.25×10^{-4}	2.98×10^{-6}
BPTT, $K = 12$	1.39×10^{-3}	8.46×10^{-5}	4.15×10^{-4}	3.02×10^{-6}
BPTT, $K = 13$	1.39×10^{-3}	8.20×10^{-5}	3.92×10^{-4}	2.89×10^{-6}
BPTT, $K = 14$	1.39×10^{-3}	8.08×10^{-5}	3.80×10^{-4}	2.86×10^{-6}
BPTT, $K = 15$	1.39×10^{-3}	8.03×10^{-5}	3.70×10^{-4}	2.85×10^{-6}

Table 2.3. Comparison of estimation accuracy, ($\sigma_e = 0.02$, $SNR = 17.8$)

Algorithm	$J_{60000}(60000)$	$J_{2000}(60000)$	$F(60000)$	$P(60000)$
BPP	5.99×10^{-3}	1.72×10^{-3}	6.01×10^{-3}	2.15×10^{-3}
SM	1.93×10^{-3}	8.75×10^{-5}	2.30×10^{-4}	1.32×10^{-5}
BPTT, $K = 1$	2.66×10^{-3}	8.97×10^{-4}	1.04×10^{-3}	7.77×10^{-5}
BPTT, $K = 2$	1.82×10^{-3}	5.71×10^{-4}	2.22×10^{-4}	5.84×10^{-6}
BPTT, $K = 3$	1.67×10^{-3}	5.32×10^{-4}	1.87×10^{-4}	2.80×10^{-6}
BPTT, $K = 4$	1.67×10^{-3}	5.31×10^{-4}	9.49×10^{-5}	1.97×10^{-6}
BPTT, $K = 5$	1.72×10^{-3}	5.45×10^{-4}	6.32×10^{-5}	1.45×10^{-6}
BPTT, $K = 6$	1.75×10^{-3}	5.59×10^{-4}	5.68×10^{-5}	5.93×10^{-6}
BPTT, $K = 7$	1.77×10^{-3}	5.82×10^{-4}	6.23×10^{-5}	4.49×10^{-6}
BPTT, $K = 8$	1.81×10^{-3}	6.04×10^{-4}	9.88×10^{-5}	5.23×10^{-6}
BPTT, $K = 9$	1.86×10^{-3}	6.15×10^{-4}	1.04×10^{-4}	1.04×10^{-5}
BPTT, $K = 10$	1.90×10^{-3}	7.68×10^{-4}	2.82×10^{-4}	1.17×10^{-5}
BPTT, $K = 11$	1.91×10^{-3}	8.61×10^{-4}	3.23×10^{-4}	1.26×10^{-5}
BPTT, $K = 12$	1.91×10^{-3}	9.24×10^{-4}	2.92×10^{-4}	1.36×10^{-5}
BPTT, $K = 13$	1.92×10^{-3}	9.94×10^{-4}	2.62×10^{-4}	1.52×10^{-5}
BPTT, $K = 14$	1.93×10^{-3}	1.07×10^{-3}	2.37×10^{-4}	1.70×10^{-5}
BPTT, $K = 15$	1.93×10^{-3}	1.15×10^{-3}	2.19×10^{-4}	1.85×10^{-5}

Table 2.4. Comparison of estimation accuracy, ($\sigma_e = 0.1$, $SNR = 3.56$)

Algorithm	$J_{60000}(60000)$	$J_{2000}(60000)$	$F(60000)$	$P(60000)$
BPP	1.94×10^{-2}	1.55×10^{-2}	2.10×10^{-2}	2.16×10^{-2}
SM	1.30×10^{-2}	1.09×10^{-2}	1.36×10^{-3}	1.07×10^{-3}
BPTT, $K = 1$	1.41×10^{-2}	1.12×10^{-2}	5.26×10^{-3}	2.79×10^{-3}
BPTT, $K = 2$	1.28×10^{-2}	1.07×10^{-2}	2.94×10^{-3}	1.18×10^{-3}
BPTT, $K = 3$	1.25×10^{-2}	1.06×10^{-2}	1.39×10^{-3}	7.23×10^{-4}
BPTT, $K = 4$	1.25×10^{-2}	1.06×10^{-2}	7.65×10^{-4}	7.93×10^{-4}
BPTT, $K = 5$	1.26×10^{-2}	1.07×10^{-2}	5.73×10^{-4}	7.55×10^{-4}
BPTT, $K = 6$	1.27×10^{-2}	1.08×10^{-2}	6.56×10^{-4}	8.89×10^{-4}
BPTT, $K = 7$	1.27×10^{-2}	1.08×10^{-2}	6.45×10^{-4}	8.69×10^{-4}
BPTT, $K = 8$	1.27×10^{-2}	1.08×10^{-2}	7.34×10^{-4}	8.83×10^{-4}
BPTT, $K = 9$	1.28×10^{-2}	1.08×10^{-2}	1.12×10^{-3}	9.23×10^{-4}
BPTT, $K = 10$	1.29×10^{-2}	1.08×10^{-2}	1.55×10^{-3}	9.69×10^{-4}
BPTT, $K = 11$	1.29×10^{-2}	1.08×10^{-2}	1.46×10^{-3}	1.01×10^{-3}
BPTT, $K = 12$	1.29×10^{-2}	1.08×10^{-2}	1.42×10^{-3}	1.05×10^{-3}
BPTT, $K = 13$	1.29×10^{-2}	1.08×10^{-2}	1.42×10^{-3}	1.04×10^{-3}
BPTT, $K = 14$	1.29×10^{-2}	1.09×10^{-2}	1.41×10^{-3}	1.05×10^{-3}
BPTT, $K = 15$	1.29×10^{-2}	1.09×10^{-2}	1.41×10^{-3}	1.06×10^{-3}

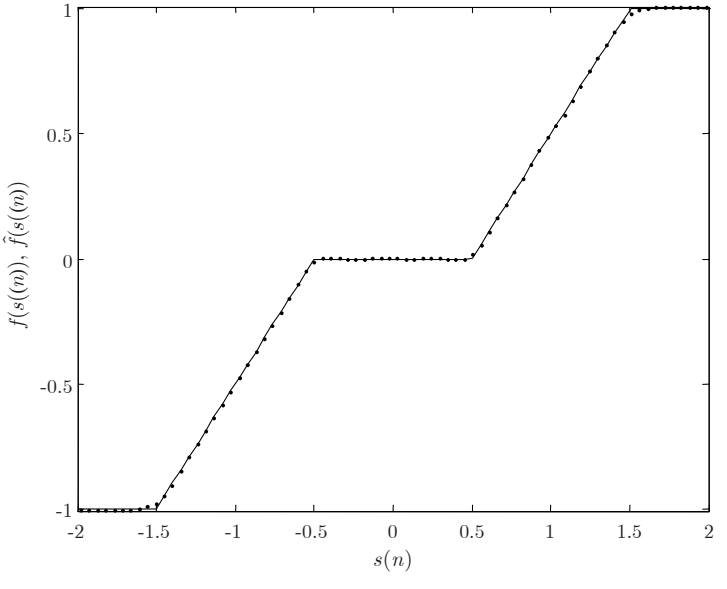

Fig. 2.10. True (solid line) and estimated (dotted line) nonlinear functions, (BPTT, $K = 4$)

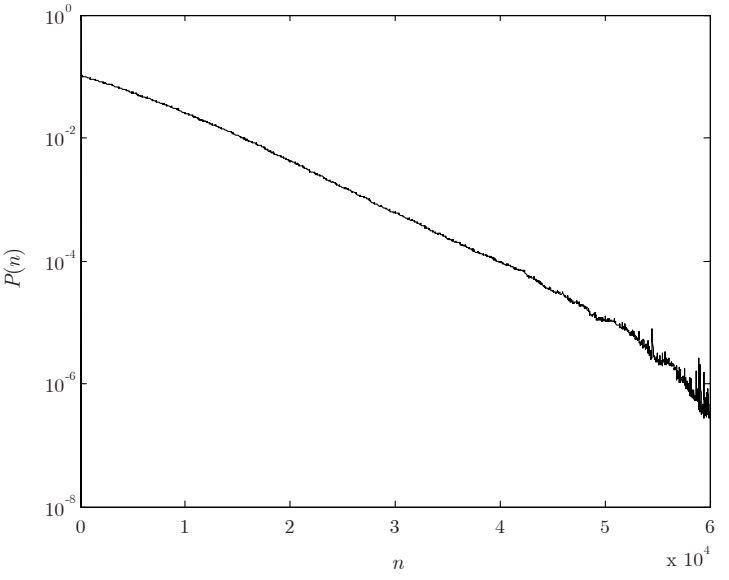

Fig. 2.11. Convergence rate of linear model parameters, (BPTT, $K = 4$)

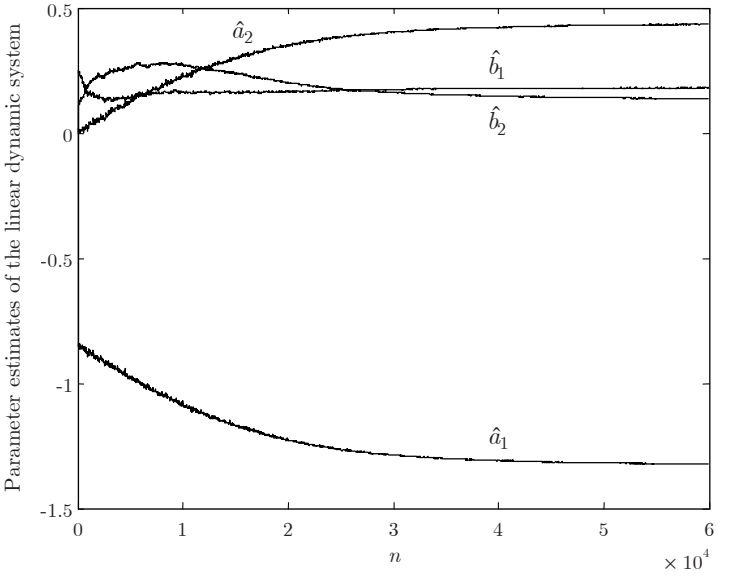

Fig. 2.12. Evolution of linear model parameters, (BPTT, $K = 4$)

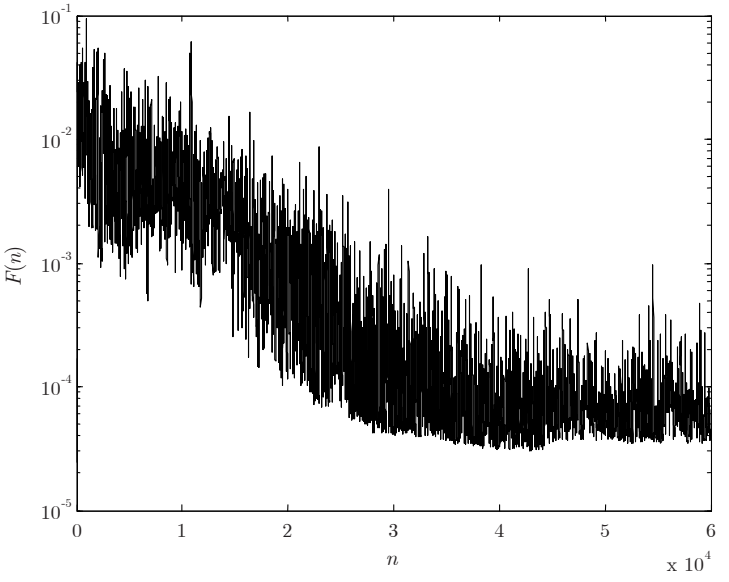

Fig. 2.13. Convergence rate of the nonlinear element model, (BPTT, $K = 4$)

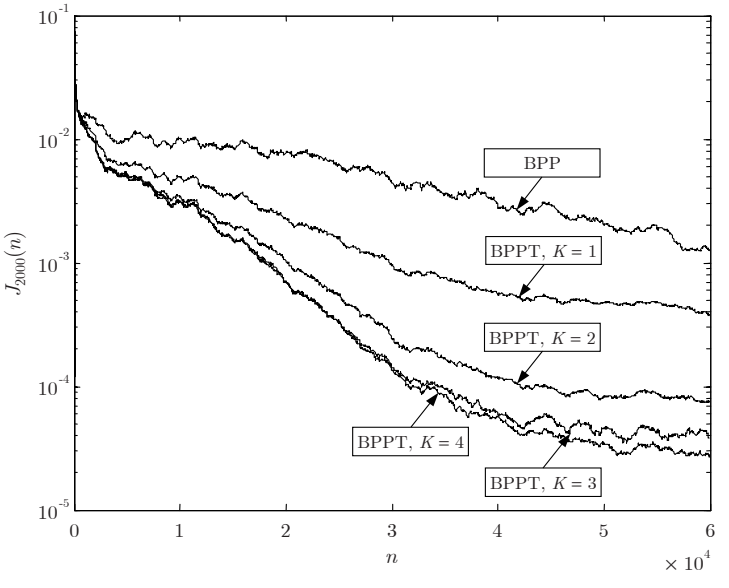

Fig. 2.14. Convergence rate of the BPP and BPPT algorithms

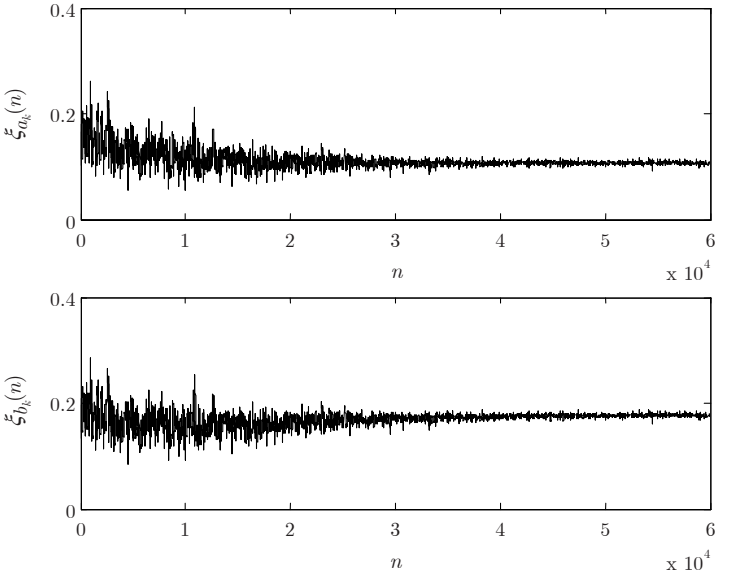

Fig. 2.15. Evolution of gradient calculation accuracy degrees, (BPTT, $K = 4$)

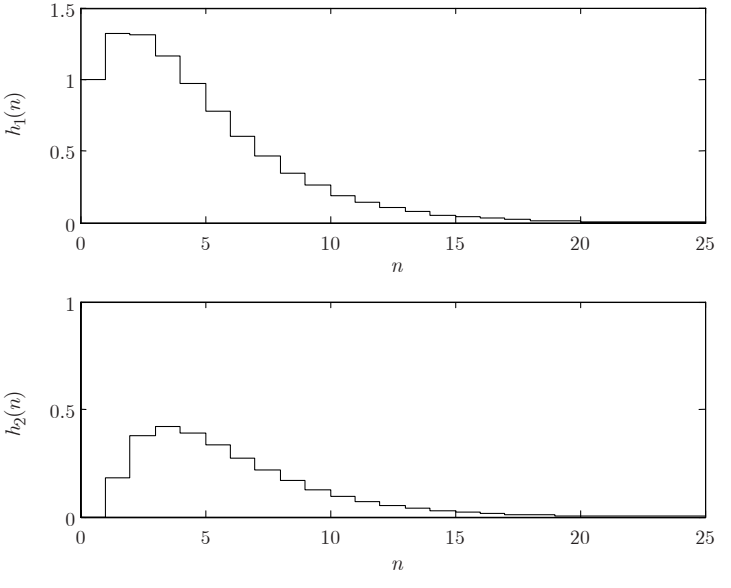

Fig. 2.16. Impulse responses of sensitivity models and the linear dynamic model, (BPTT, $K = 4$)

2.6 Two-tank system example

A laboratory two-tank system with transport delay, shown in Fig. 2.17, was used as a source of data for a real process example [89]. A system actuator, consisting of the dc motor M, the diaphragm water pump P, and the bypass valve V, was identified. The bypass valve V was left in an open position, introducing a dead zone into the system steady-state characteristic, see Fig. 2.21 for the characteristic obtained via an experiment with recording a system response to a stairs-wise input. The parameters of both a parallel neural network Wiener model and a linear autoregressive with an exogenous input (ARX) model were determined based on a sequence of 6000 input and output data recorded in another open loop experiment controlled by a computer at the sampling interval of 1s. The motor M was driven by a control signal obtained from a pseudorandom generator of uniform distribution and transformed in such a way that each pseudorandom value was kept constant for ten sampling intervals constituting ten successive control values. The flow rate of water, measured with the Venturi flow meter F, was chosen as a system output and the squared control signal was used as the model input. Before identification, the input and output data were scaled to zero mean values and variances 1.

First, the parameters of the ARX model were estimated with the least squares method. Based on the results of ARX model estimation and applying the Akaike final prediction error criterion [112], the second order model with delay 1 was chosen, i.e., $na = 2$, $nb = 2$. Then the parallel neural network Wiener model containing 30 nonlinear nodes of the hyperbolic tangent activation function was trained recursively with BPP, the SM, and truncated BPTT at $K = 1, \ldots, 15$ with the input-output data set processed cyclically five times. Finally, the trained neural network Wiener models were tested with another set of 3000 input-output data. The obtained results, i.e., the values of $J_{3000}(30000)$ for the training set and $J_{3000}(3000)$ for the testing set, are given in Table 2.5 and illustrated in Figs 2.18 – 2.21. The inspection of the impulse responses $h_1(n)$ and $h_2(n)$, presented in Figs 2.18 and 2.19, shows that both of them decrease to practically negligible small values at $n = 12$. Therefore, according to Theorems 2.1 and 2.2, no significant improvement can be expected with unfolding the model for more than 12 time steps – see the results in Table 2.5 for a confirmation of this conclusion. As can be expected, the results obtained for the Wiener model with the SM and BPTT at $K = 2, \ldots, 15$ are better than the result for the ARX model. Moreover, contrary to the simulation example, a gradual improvement in model fitting can be seen (Fig. 2.20) with an increase in K which can be explained by applying a sufficiently low learning rate of 0.001. The operation of the diaphragm water pump at constant excitation causes some periodic changes of the water flow rate. These changes can be considered as additive output disturbances and, as a result, the cost function achieves much higher values than in the case of the noise-free simulation example, see Tables 2.2 and 2.5.

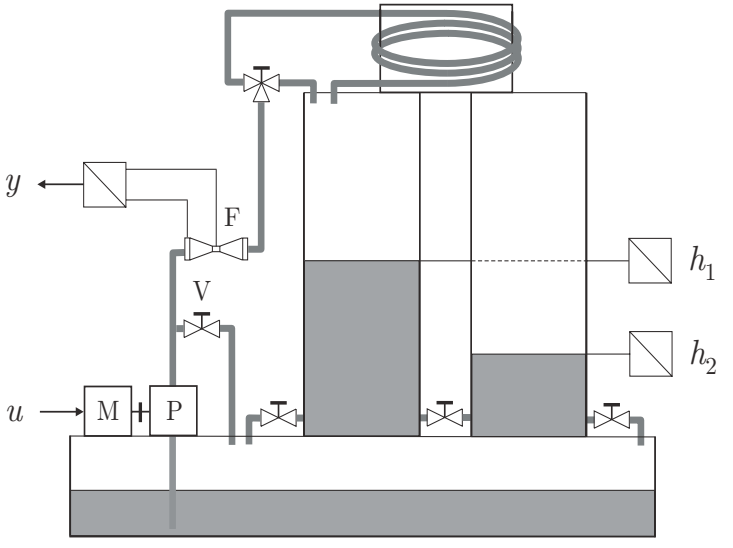

Fig. 2.17. Two-tank system with transport delay

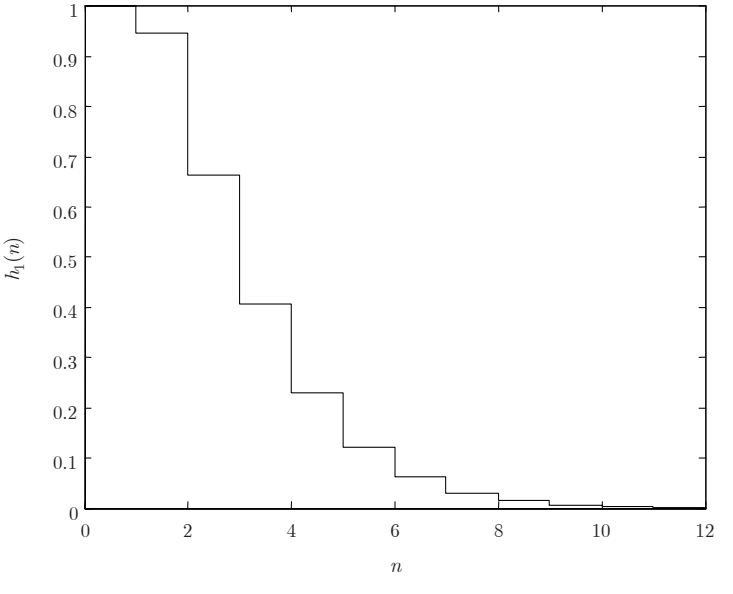

Fig. 2.18. Two-tank system. Impulse response of the system $\hat{H}_1(q^{-1})$, $K=15$

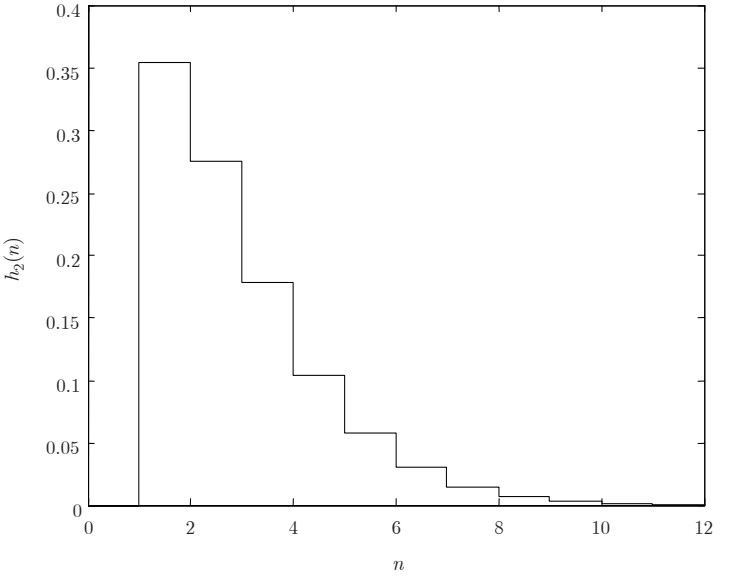

Fig. 2.19. Two-tank system. Impulse response of the system $\hat{H}_2(q^{-1})$, $K=15$

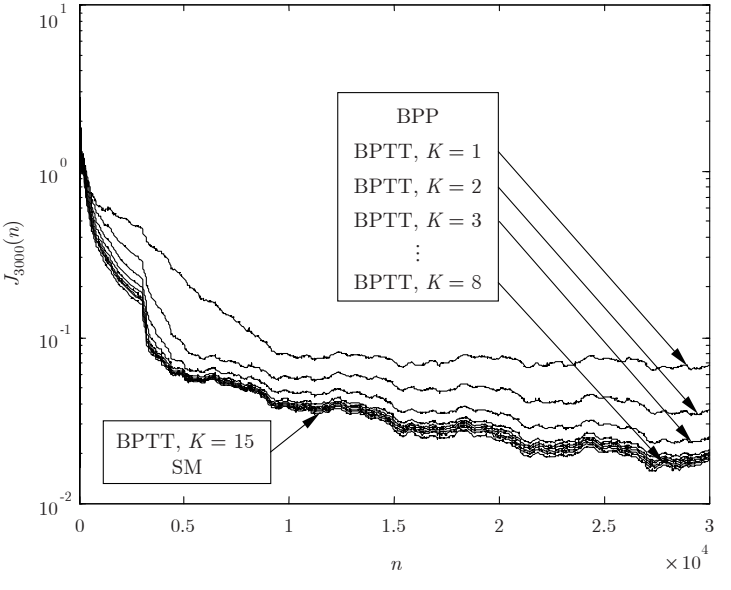

Fig. 2.20. Two-tank system. Training the Wiener model with BPP, BPTT, and the SM

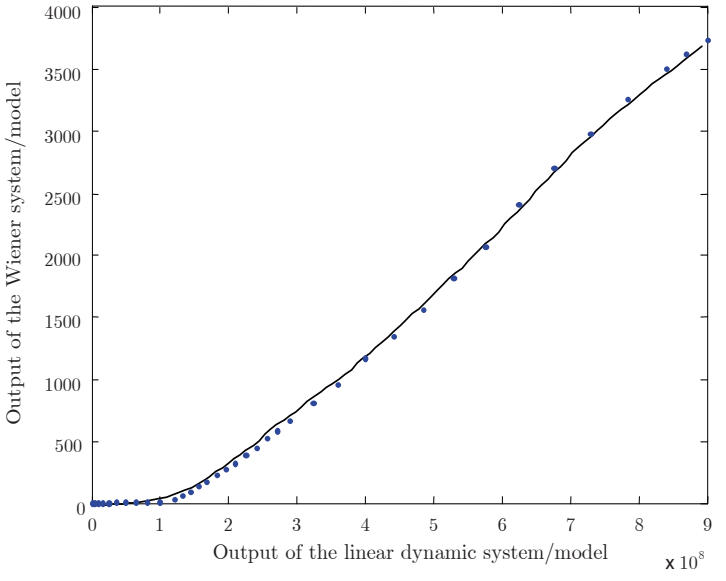

Fig. 2.21. Two-tank system. The true (dotted line) and estimated (solid line) nonlinear characteristics, BPTT, $K = 15$

Table 2.5. Two-tank system. Comparison of estimation accuracy

Algorithm	$J_{3000}(30000)$	$J_{3000}(3000)$
ARX		0.0406
BPP	0.0690	0.0860
SM	0.0176	0.0196
BPTT, $K = 1$	0.0374	0.0461
BPTT, $K = 2$	0.0253	0.0306
BPTT, $K = 3$	0.0210	0.0246
BPTT, $K = 4$	0.0200	0.0229
BPTT, $K = 5$	0.0194	0.0221
BPTT, $K = 6$	0.0188	0.0213
BPTT, $K = 7$	0.0183	0.0206
BPTT, $K = 8$	0.0181	0.0202
BPTT, $K = 9$	0.0179	0.0200
BPTT, $K = 10$	0.0178	0.0199
BPTT, $K = 11$	0.0178	0.0198
BPTT, $K = 12$	0.0178	0.0198
BPTT, $K = 13$	0.0178	0.0197
BPTT, $K = 14$	0.0177	0.0197
BPTT, $K = 15$	0.0177	0.0197

It is well known that parallel models are more suitable for the identification of noise-corrupted systems as they do not use past system outputs, which are noise-corrupted, to calculate the model output. Clearly, the parallel neural network Wiener model is also well suited for the identification of Wiener systems with additive white output noise. On the other hand, from the fact that series-parallel models use past system outputs, it follows that these models are not able to describe systems with additive white output noise appropriately, and the series-parallel neural network Wiener model is not suitable for the identification of Wiener systems additive disturbances either. The noise interfering with the system makes the identification a stochastic optimization problem.
Therefore, a large number of input-output data should be used to achieve a required level of model accuracy for systems with a high level of output disturbances. Moreover, small or slowly decreasing values of the learning rate η are necessary to achieve the convergence of steepest descent algorithms.

2.7 Prediction error method

It is well known that steepest descent algorithms have a linear convergence rate and can be quite slow in many practical cases. A faster convergence rate can be achieved with methods which are based on a second-order approximation of the optimization criterion to determine the search direction. An alternative to steepest descent methods is the prediction error (PE) method or its pattern version – the recursive prediction error (RPE) method. The PE and RPE algorithms have superior convergence properties in comparison with steepest descent algorithms as they use the approximate Hessjan to compute the search direction. The properties of the PE and RPE algorithms for different neural network models are discussed in [29, 127]. RPE algorithms for Wiener models with known static nonlinearity or the nonlinear model approximated with a piecewise linear function were analyzed by Wigren [165, 166].
A batch PE algorithm for Wiener models with a polynomial model of the nonlinear element was used by Norquay *et al.* [128]. A sequential version of the PE algorithm for the training of a SISO neural network Wiener model was proposed in [85]. In both these algorithms, the gradient of the model output w.r.t. the parameters of its linear part is computed with the SM.

2.7.1 Recursive prediction error learning algorithm

The identification problem can be formulated as follows: Given a set of input and output data $\{Z^N = \{u(n), y(n)\},\ n = 1, \ldots, N,\}$ and a candidate neural network Wiener model, estimate the parameters $\boldsymbol{\theta}$ of the model so that the predictions $\hat{y}(n|n-1)$ of the system output are close to the system output $y(n)$ in the sense of the following mean square error criterion:

$$J(\boldsymbol{\theta}, Z^N) = \frac{1}{2N}\sum_{n=1}^{N}\big(y(n) - \hat{y}(n|n-1)\big)^2. \tag{2.94}$$

A solution to this problem, for nonlinearly parameterized models, is a gradient technique based on the approximation of the Hessjan known as the prediction error method. The PE method is a batch optimization method of the Gauss-Newton type. The RPE algorithm is a recursive counterpart of the PE method. Both the PE and RPE algorithms are guaranteed to converge to a local minimum of $J(\boldsymbol{\theta}, Z^N)$ with the probability 1 as $N \to \infty$ [29]. Given the gradient

$$\begin{aligned}\boldsymbol{\psi}(n) = \Bigg[&\frac{\partial\hat{y}(n|n-1)}{\partial\hat{a}_1}\cdots\frac{\partial\hat{y}(n|n-1)}{\partial\hat{a}_{na}}\ \frac{\partial\hat{y}(n|n-1)}{\partial\hat{b}_1}\cdots\frac{\partial\hat{y}(n|n-1)}{\partial\hat{b}_{nb}}\\ &\frac{\partial\hat{y}(n|n-1)}{\partial w_{10}^{(1)}}\cdots\frac{\partial\hat{y}(n|n-1)}{\partial w_{M1}^{(1)}}\ \frac{\partial\hat{y}(n|n-1)}{\partial w_{10}^{(2)}}\cdots\frac{\partial\hat{y}(n|n-1)}{\partial w_{1M}^{(2)}}\Bigg]^T\end{aligned} \tag{2.95}$$

of the model output w.r.t. the parameter vector

$$\boldsymbol{\theta} = \big[\hat{a}_1\ldots\hat{a}_{na}\ \hat{b}_1\ldots\hat{b}_{nb}\ w_{10}^{(1)}\ldots w_{M1}^{(1)} w_{10}^{(2)}\ldots w_{1M}^{(2)}\big]^T, \tag{2.96}$$

the RPE algorithm can be expressed as [29, 127]:

$$\mathbf{K}(n) = \frac{\mathbf{P}(n-1)\boldsymbol{\psi}(n)}{1+\boldsymbol{\psi}^T(n)\mathbf{P}(n-1)\boldsymbol{\psi}(n)}, \tag{2.97}$$

$$\boldsymbol{\theta}(n) = \boldsymbol{\theta}(n-1) + \mathbf{K}(n)\big(y(n) - \hat{y}(n|n-1)\big), \tag{2.98}$$

$$\mathbf{P}(n) = \mathbf{P}(n-1) - \mathbf{K}(n)\boldsymbol{\psi}^T(n)\mathbf{P}(n-1). \tag{2.99}$$

The RPE algorithm is an alternative to the well-known steepest descent methods and has much better convergence properties and lower memory requirements. Computational complexity of the RPE algorithm is much higher and it is not recommended for problems with a large number of estimated parameters. Note that replacing $\mathbf{K}(n)$ with $\eta\boldsymbol{\psi}(n)$ in (2.98), where η denotes the learning rate, results in the steepest descent algorithm.
In some cases, it may be useful to apply algorithms that combine quasi-Newton methods and the steepest descent approach. In the RPE and SM algorithm, used in the example below for comparison, a search direction is calculated as a sum of the search directions of the RPE method and the SM.

2.7.2 Pneumatic valve simulation example

As a source of data for testing the RPE algorithm and comparing it with the SM, and the RPE and SM algotithm, a model of a pneumatic valve was used [165]. The dynamic balance between the pneumatic control signal $u(n)$ applied

to the valve stem, a counteractive spring force and friction is described by the linear dynamics

$$\begin{aligned} s(n) = {} & 1.4138s(n-1) - 0.6065s(n-2) + 0.1044u(n-1) \\ & + 0.0833u(n-2). \end{aligned} \tag{2.100}$$

The flow through the valve is a nonlinear function of the valve stem position $s(n)$:

$$f\big(s(n)\big) = \frac{0.3163s(n)}{\sqrt{0.1 + 0.9s^2(n)}}. \tag{2.101}$$

It is assumed that the model output is additively disturbed by a zero-mean discrete white Gaussian noise $\varepsilon(n)$ with the standard deviation $\sigma_\varepsilon = 0.01$:

$$y(n) = f\big(s(n)\big) + \varepsilon(n). \tag{2.102}$$

A sequence of 10000 pseudorandom numbers, uniformly distributed in $(-1, 1)$, was used as the model input. Based on the simulated input-output data, the parallel neural network Wiener model containing 10 nonlinear nodes of the hyperbolic tangent activation function was trained with the RPE, SM, and RPE and SM algorithms.
The identification results are illustrated in Figs 2.22 – 2.25 [87]. To compare estimation accuracy of linear system parameters and the nonlinear function $f(\cdot)$, the indices (2.91) and (2.92) are used.

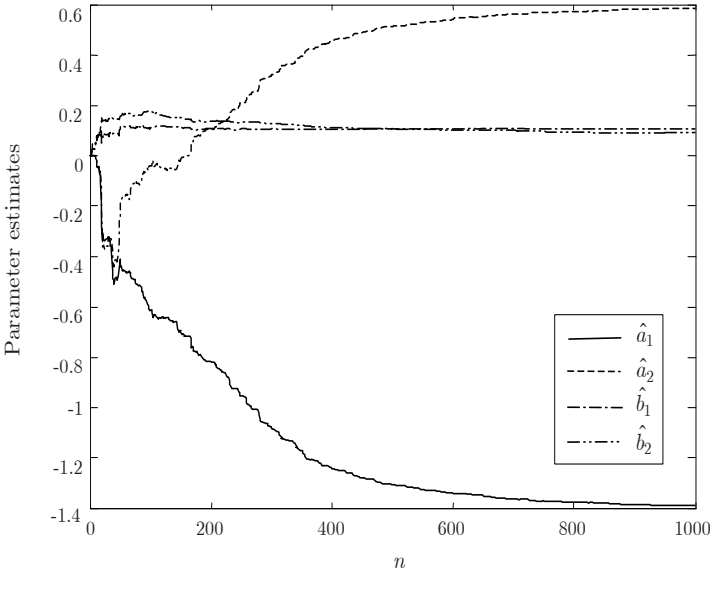

Fig. 2.22. RPE algorithm. Evolution of parameter estimates of the linear element

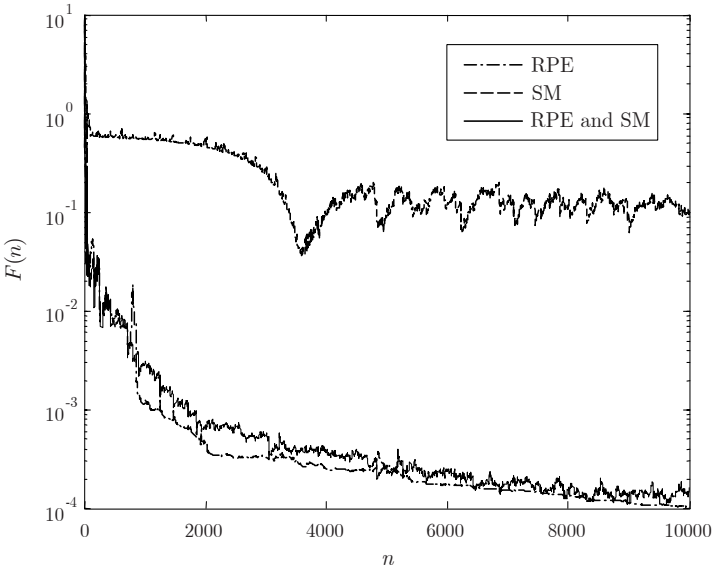

Fig. 2.23. Convergence of the linear element model

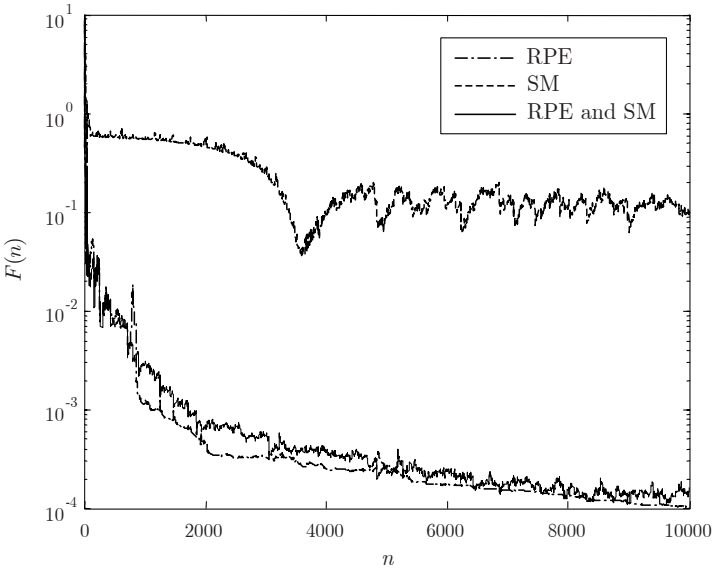

Fig. 2.24. Convergence of the nonlinear element model

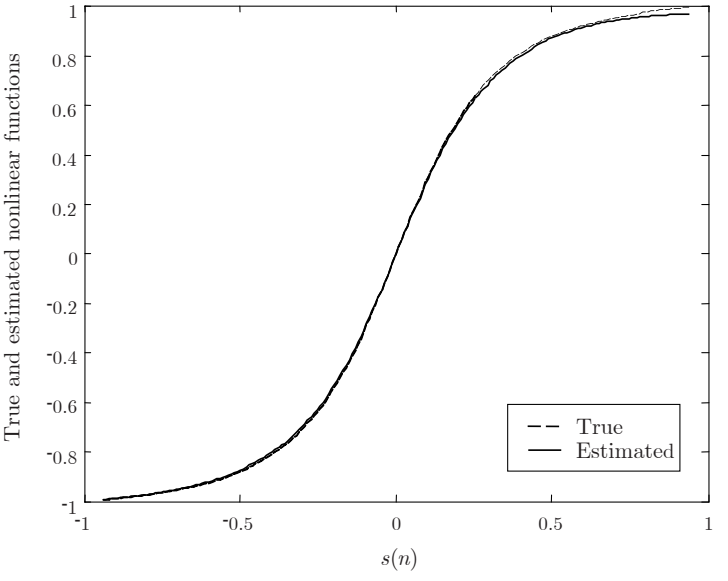

Fig. 2.25. RPE algorithm. True and estimated nonlinear characteristics

2.8 Summary

In this chapter, various steepest descent learning algorithms for neural network Wiener models have been derived, analyzed and compared in a unified framework. For the truncated BPTT method, it has been shown that the accuracy of gradient calculation depends on impulse responses of the linear dynamic model and its sensitivity models. The number of unfolded time steps can be determined on the basis of the number of time steps necessary for the impulse response of sensitivity models to decrease to negligible small values. The application of these algorithms to both simulated and real data systems has been demonstrated as well.

The parallel neural network Wiener model is simpler than its series-parallel counterpart as it does not contain the inverse model of the static nonlinear element. Another important advantage of the parallel neural network model is that its training with the BPP algorithm requires almost half the computational burden for the training of the series-parallel model with the BPS algorithm. Applying the SM or truncated BPTT algorithms, one can expect a higher convergence rate, in comparison with the BPP algorithm, as a more accurate gradient approximation is used. In comparison with gradient computation in feedforward neural networks, gradient computation in recurrent networks is commonly much more computationally intensive. That is not the case for the parallel neural network Wiener model, as both the SM and truncated BPTT algorithms require only a little more computational effort than

the BPP algorithm. This comes from the fact that, in all these algorithms, $3M + 1$ parameters of the nonlinear element model are adjusted in the same way. The algorithms update the parameters of the linear dynamic model in different manners but the number of the parameters $na + nb$ is commonly not greater than a few.

Both the simulation example and the real process example show the lowest convergence rate of the BPP algorithm. In the simulation examples, the highest accuracy of both the linear dynamic model and the nonlinear element has been obtained using the truncated BPTT algorithm and unfolding the linear dynamic model only a few steps back in time.

The RPE identification algorithm has been also extended to training recurrent neural network Wiener models. The calculation of the gradient is performed with the SM and it does not require much more computation than the BPP algorithm. In comparison with steepest descent methods, the RPE algorithm has much better convergence properties at the expense of higher computational complexity. The RPE algorithm can be very useful when the system output is disturbed additively by the white noise. Note that in such cases, the steepest descent algorithms require a small step size to achieve convergence to a local minima of the performance function.

Although only sequential learning versions of the basic steepest descent algorithm for neural network Wiener models are described and discussed in this chapter, their batch counterparts can be derived easily given the rules for gradient computation. Also, the implementation of other gradient methods such as variable metric methods, conjugate gradient methods, or the RLS learning algorithms (a review and discussion on their hardware implementation using the systolic technology is given by Rutkowski [144]) is straightforward.

2.9 Appendix 2.1. Gradient derivation of truncated BPTT. SISO Wiener models

Assume that the parallel SISO Wiener model is K times unfolded back in time. Let $\partial^+\hat{s}(n)/\partial\hat{a}_k$ and $\partial^+\hat{s}(n)/\partial\hat{b}_k$ denote partial derivatives calculated for the model unfolded back in time for K time steps, and $\partial\hat{s}(n)/\partial\hat{a}_k$ and $\partial\hat{s}(n)/\partial\hat{b}_k$ – partial derivatives calculated without taking into account the dependence of past model outputs on $\hat{a}_k$ and $\hat{b}_k$, respectively. The differentiation of (2.15) gives

$$\begin{aligned}\frac{\partial^+\hat{s}(n)}{\partial\hat{a}_k} &= \frac{\partial\hat{s}(n)}{\partial\hat{a}_k} + \sum_{i_1=1}^{K}\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)}\frac{\partial\hat{s}(n-i_1)}{\partial\hat{a}_k} \\ &= -\hat{s}(n-k) - \sum_{i_1=1}^{K}\hat{s}(n-k-i_1)\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)},\end{aligned} \tag{2.103}$$

where $k = 1, \ldots, na$,

$$\begin{aligned}\frac{\partial^+\hat{s}(n)}{\partial\hat{b}_k} &= \frac{\partial\hat{s}(n)}{\partial\hat{b}_k} + \sum_{i_1=1}^{K}\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)}\frac{\partial\hat{s}(n-i_1)}{\partial\hat{b}_k} \\ &= u(n-k) + \sum_{i_1=1}^{K}u(n-k-i_1)\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)},\end{aligned} \tag{2.104}$$

where $k = 1, \ldots, nb$. From (2.15), it follows that partial derivatives of the actual output of the linear dynamic model $\hat{s}(n)$ w.r.t. the delayed outputs $\hat{s}(n-1), \ldots, \hat{s}(n-K)$ are

$$\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)} = -\sum_{m=1}^{na}\hat{a}_m\frac{\partial\hat{s}(n-m)}{\partial\hat{s}(n-i_1)} = -\sum_{m=1}^{na}\hat{a}_m\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1+m)}, \tag{2.105}$$

where $i_1 = 1, \ldots, K$, and $\partial\hat{s}(n-m)/\partial\hat{s}(n-i_1) = 0$ for $m > i_1$. Finally, to obtain the partial derivatives $\partial\hat{y}(n)/\partial\hat{a}_k$ and $\partial\hat{y}(n)/\partial\hat{b}_k$, the partial derivatives $\partial\hat{s}(n)/\partial\hat{a}_k$ and $\partial\hat{s}(n)/\partial\hat{b}_k$ in (2.48) and (2.49) are replaced with $\partial^+\hat{s}(n)/\partial\hat{a}_k$ and $\partial^+\hat{s}(n)/\partial\hat{b}_k$:

$$\frac{\partial\hat{y}(n)}{\partial\hat{a}_k} = \left(-\hat{s}(n-k) - \sum_{i_1=1}^{K}\hat{s}(n-k-i_1)\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)}\right)\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}, \tag{2.106}$$

$$\frac{\partial\hat{y}(n)}{\partial\hat{b}_k} = \left(u(n-k) + \sum_{i_1=1}^{K}u(n-k-i_1)\frac{\partial\hat{s}(n)}{\partial\hat{s}(n-i_1)}\right)\frac{\partial\hat{y}(n)}{\partial\hat{s}(n)}. \tag{2.107}$$

2.10 Appendix 2.2. Gradient derivation of truncated BPTT. MIMO Wiener models

Assume that the parallel MIMO Wiener model is K times unfolded back in time. Let $\partial^+\hat{s}_l(n)/\partial\hat{a}_{rp}^{(k)}$ and $\partial^+\hat{s}_l(n)/\partial\hat{b}_{rp}^{(k)}$ denote partial derivatives calculated for the model unfolded back in time for K time steps, and $\partial\hat{s}_l(n)/\partial\hat{a}_{rp}^{(k)}$ and $\partial\hat{s}_l(n)/\partial\hat{b}_{rp}^{(k)}$ – partial derivatives calculated without taking into account the dependence of past model outputs on $\hat{a}_{rp}^{(k)}$ and $\hat{b}_{rp}^{(k)}$, respectively. The differentiation of (2.71) gives

$$\begin{aligned}\frac{\partial^+\hat{s}_l(n)}{\partial\hat{a}_{rp}^{(k)}} &= \frac{\partial\hat{s}_l(n)}{\partial\hat{a}_{rp}^{(k)}} + \sum_{i_1=1}^{K}\sum_{i_2=1}^{ns}\frac{\partial\hat{s}_l(n)}{\partial\hat{s}_{i_2}(n-i_1)}\frac{\partial\hat{s}_{i_2}(n-i_1)}{\partial\hat{a}_{rp}^{(k)}} \\ &= -\delta_{lr}\hat{s}_p(n-k) - \sum_{i_1=1}^{K}\hat{s}_p(n-k-i_1)\frac{\partial\hat{s}_l(n)}{\partial\hat{s}_r(n-i_1)},\end{aligned} \tag{2.108}$$

where

$$\delta_{lr} = \begin{cases} 1 & \text{for } l = r \\ 0 & \text{for } l \neq r \end{cases}, \tag{2.109}$$

where $l = 1,\ldots,ns$, $r = 1,\ldots,ns$, $p = 1,\ldots,ns$, $k = 1,\ldots,na$,

$$\begin{aligned}\frac{\partial^+\hat{s}_l(n)}{\partial\hat{b}_{rp}^{(k)}} &= \frac{\partial\hat{s}_l(n)}{\partial\hat{b}_{rp}^{(k)}} + \sum_{i_1=1}^{K}\sum_{i_2=1}^{ns}\frac{\partial\hat{s}_l(n)}{\partial\hat{s}_{i_2}(n-i_1)}\frac{\partial\hat{s}_{i_2}(n-i_1)}{\partial\hat{b}_{rp}^{(k)}} \\ &= \delta_{lr}u_p(n-k) + \sum_{i_1=1}^{K}u_p(n-k-i_1)\frac{\partial\hat{s}_l(n)}{\partial\hat{s}_r(n-i_1)},\end{aligned} \tag{2.110}$$

where $l = 1,\ldots,ns$, $r = 1,\ldots,ns$, $p = 1,\ldots,nu$, $k = 1,\ldots,nb$. From (2.71), it follows that partial derivatives of the actual lth output of the linear dynamic model w.r.t. its delayed rth output are

$$\frac{\partial\hat{s}_l(n)}{\partial\hat{s}_r(n-i_1)} = -\sum_{m=1}^{na}\sum_{m_1=1}^{ns}\hat{a}_{lm_1}^{(m)}\frac{\partial\hat{s}_{m_1}(n-m)}{\partial\hat{s}_r(n-i_1)} = -\sum_{m=1}^{na}\hat{a}_{lm_1}^{(m)}\frac{\partial\hat{s}_{m_1}(n)}{\partial\hat{s}_r(n-i_1+m)}, \tag{2.111}$$

where

$$\frac{\partial\hat{s}_{m_1}(n-m)}{\partial\hat{s}_r(n-i_1)} = 0 \text{ for } m > i_1 \text{ or } m = i_1 \text{ and } m_1 \neq r. \tag{2.112}$$

Finally, partial derivatives of the tth model output w.r.t. the parameters $\hat{a}_{rp}^{(k)}$ and $\hat{b}_{rp}^{(k)}$ are

$$\frac{\partial\hat{y}_t(n)}{\partial\hat{a}_{rp}^{(k)}} = \left(-\delta_{lr}\hat{s}_p(n-k) - \sum_{i_1=1}^{K}\hat{s}_p(n-k-i_1)\frac{\partial\hat{s}_l(n)}{\partial\hat{s}_r(n-i_1)}\right)\frac{\partial\hat{y}_t(n)}{\hat{s}_l(n)}, \tag{2.113}$$

$$\frac{\partial \hat{y}_t(n)}{\partial \hat{b}_{rp}^{(k)}} = \left(\delta_{lr} u_p(n-k) + \sum_{i_1=1}^{K} u_p(n-k-i_1) \frac{\partial \hat{s}_l(n)}{\partial \hat{s}_r(n-i_1)}\right) \frac{\partial \hat{y}_t(n)}{\hat{s}_l(n)}, \qquad (2.114)$$

where $\partial \hat{y}_t(n)/\hat{s}_l(n)$ is given by (2.65).

2.11 Appendix 2.3. Proof of Theorem 2.1

In the BPTT method, the Wiener model is unfolded back in time and then the unfolded model is differentiated w.r.t. model parameters. This is equivalent to the differentiation of the model and unfolding the differentiated model, i.e., sensitivity models back in time. Taking into account (2.77), and (2.78), from (2.52) it follows that the outputs of sensitivity models for the parameters $\hat{a}_k$, $k = 1, \ldots, na$, can be expressed as functions of the input $u(n)$:

$$\begin{aligned} \frac{\partial \hat{s}(n)}{\partial \hat{a}_k} &= -H_1(q^{-1})\hat{s}(n-k) = -H_1(q^{-1})H_2(q^{-1})u(n-k) \\ &= -H(q^{-1})u(n-k), \end{aligned} \qquad (2.115)$$

where

$$H(q^{-1}) = H_1(q^{-1})H_2(q^{-1}). \qquad (2.116)$$

The impulse response of the system (2.116) is given by the convolution relationship

$$h(n) = \sum_{i_2=0}^{n} h_1(i_2)h_2(n-i_2). \qquad (2.117)$$

Therefore, the partial derivative $\partial \hat{s}(n)/\partial \hat{a}_k$ is related to the input $u(n)$ with the impulse response functions $h_1(n)$ and $h_2(n)$:

$$\frac{\partial \hat{s}(n)}{\partial \hat{a}_k} = -\sum_{i_1=0}^{n-k} h(i_1)u(n-k-i_1) = -\sum_{i_1=0}^{n-k} \sum_{i_2=0}^{i_1} h_1(i_2)h_2(i_1-i_2)u(n-k-i_1). \qquad (2.118)$$

The calculation of (2.118) requires unfolding sensitivity models $n-1$ times back in time. Thus, the partial derivatives $\partial^+ \hat{s}(n)/\partial \hat{a}_k$ calculated for sensitivity models unfolded K times back in time are

$$\frac{\partial^+ \hat{s}(n)}{\partial \hat{a}_k} = \begin{cases} -\displaystyle\sum_{i_1=0}^{n-k} \sum_{i_2=0}^{i_1} h_1(i_2)h_2(i_1-i_2)u(n-k-i_1), & \text{for } n \leqslant K+k \\ -\displaystyle\sum_{i_1=0}^{K} \sum_{i_2=0}^{i_1} h_1(i_2)h_2(i_1-i_2)u(n-k-i_1) & \\ \quad -\displaystyle\sum_{i_1=K+1}^{n-k} \sum_{i_2=0}^{K} h_1(i_2)h_2(i_1-i_2)u(n-k-i_1), & \text{for } n > K+k. \end{cases} \qquad (2.119)$$

From (2.119) it follows that, the errors of computing partial derivatives with the truncated BPTT method are

$$\Delta\hat{s}_{\hat{a}_k}(n) = \begin{cases} 0, & \text{for } n \leqslant K+k \\ -\sum\limits_{i_1=K+1}^{n-k} \sum\limits_{i_2=K+1}^{i_1} h_1(i_2)h_2(i_1-i_2)u(n-k-i_1), & \text{for } n > K+k. \end{cases} \tag{2.120}$$

Since $\{u(n)\}$ is a sequence of zero-mean i.i.d. random variables, $\Delta\hat{s}_{\hat{a}_k}(n)$ is a weighted sum of $n-k-K$ zero-mean i.i.d. random variables. Therefore, the variances of the errors (2.120) and are given by (2.76) for $n > K+k$, and are equal to zero otherwise.

2.12 Appendix 2.4. Proof of Theorem 2.2

In the BPTT method, the Wiener model is unfolded back in time and then the unfolded model is differentiated w.r.t. model parameters. This is equivalent to the differentiation of the model and unfolding the differentiated model, i.e., sensitivity models back in time. Taking into account (2.80), from (2.53) it follows that the outputs of sensitivity models for the parameters $\hat{b}_k$, $k = 1, \ldots, nb$, can be expressed as functions of the input $u(n)$:

$$\frac{\partial\hat{s}(n)}{\partial\hat{b}_k} = H_1(q^{-1})u(n-k). \tag{2.121}$$

The partial derivatives $\partial\hat{s}(n)/\partial\hat{b}_k$ are related to the input $u(n)$ with the impulse response function $h_1(n)$:

$$\frac{\partial\hat{s}(n)}{\partial\hat{b}_k} = \sum_{i_1=0}^{n-k} h_1(i_1)u(n-k-i_1). \tag{2.122}$$

The calculation of (2.122) requires unfolding sensitivity models $n-1$ times back in time. Thus, the partial derivatives $\partial^+\hat{s}(n)/\partial\hat{b}_k$ calculated for sensitivity models unfolded K times back in time are

$$\frac{\partial^+\hat{s}(n)}{\partial\hat{b}_k} = \begin{cases} \sum\limits_{i_1=0}^{n-k} h_1(i_1)u(n-k-i_1), & \text{for } n \leqslant K+k \\ \sum\limits_{i_1=0}^{K} h_1(i_1)u(n-k-i_1), & \text{for } n > K+k. \end{cases} \tag{2.123}$$

Therefore, the errors of computing partial derivatives with the truncated BPTT method are

$$\Delta\hat{s}_{\hat{b}_k}(n) = \begin{cases} 0, & \text{for } n \leqslant K+k \\ \displaystyle\sum_{i_1=K+1}^{n-k} h_1(i_1)u(n-k-i_1), & \text{for } n > K+k. \end{cases} \tag{2.124}$$

Since $\{u(n)\}$ is a sequence of zero-mean i.i.d. random variables, $\Delta\hat{s}_{\hat{b}_k}(n)$ is a weighted sum of $n-k-K$ zero-mean i.i.d. random variables. Therefore, the variances of the errors (2.124) are given by (2.79) for $n > K+k$, and are equal to zero otherwise.