11

NONLINEAR ADAPTIVE FILTERING

11.1 INTRODUCTION

The classic adaptive-filtering algorithms, such as those discussed in the remaining chapters of this book, consist of adapting the coefficients of linear filters in real time. These algorithms have applications in a number of situations where the signals measured in the environment can be well modeled as Gaussian noises applied to linear systems, and their combinations are of additive type. In digital communication systems, most of the classical approaches model the major impairment affecting the transmission with a linear model. For example, channel noise is considered additive Gaussian noise, intersymbol and co-channel interferences are also considered of additive type, and channel models are assumed to be linear frequency selective filters. While these models are accurate, there is nothing wrong with the use of linear adaptive filters[1] to remedy these impairments. However, the current demand for higher-speed communications leads to the exploration of the channel resources beyond the range their models can be considered linear. For example, when the channel is the pair of wires of the telephone system, it is widely accepted that linear models are not valid for data transmission above 4.8 Kb/s. Signal companding, amplifier saturation, multiplicative interaction between Gaussian signals, and nonlinear filtering of Gaussian signals are typical phenomena occurring in communication systems that cannot be well modeled with linear adaptive systems. In addition, if the channel transfer function does not have minimum phase and/or the signal to noise ratio is not high enough, the use of linear adaptive-filtering equalizer yields poor performance measured in terms of bit error rate. A major drawback of dealing with nonlinear models is the lack of mathematical tools that, on the other hand, are widely available for linear models. The lack of analytical tools originates in the high degrees and dimensionality of the nonlinearities. The improved performance of the nonlinear equalizer is mainly justified by extensive simulation results available in the literature, where the bit error rate is used as a performance measure.

In this chapter, we will describe some of the techniques available to model nonlinear systems using nonlinear adaptive systems using the general structure depicted in Fig. 11.1. In particular, the following approaches for nonlinear adaptive filtering will be discussed here:

[1]The reader should bear in mind that adaptive filters are nonlinear filters, even if we are adapting the coefficients of a linear filter structure, therefore the term linear adaptive filter means that we are adapting the coefficients of a linear filter structure.

P.S.R. Diniz, *Adaptive Filtering*, DOI: 10.1007/978-0-387-68606-6_11,

1. The nonrecursive polynomial model based on the Volterra series expansion.
2. The recursive polynomial model based on nonlinear difference equations.
3. The multilayer perceptron (MLP) neural network.
4. The radial basis function (RBF) neural network.

In the following sections, we will introduce the methods above mentioned for modeling nonlinear systems and for each approach adaptive algorithms for updating the corresponding nonlinear filter coefficients will be described. The chapter includes examples aimed at comparing the different structures and algorithms.

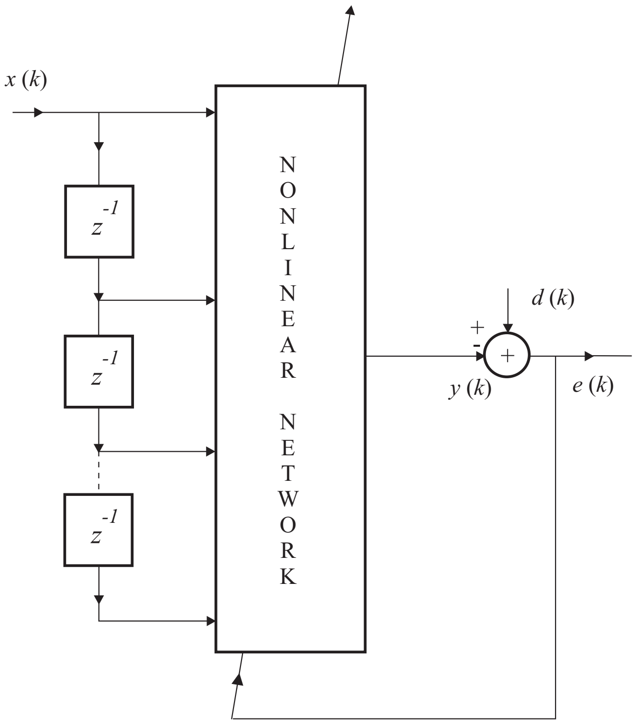


Figure 11.1 Adaptive nonlinear filter.

11.2 THE VOLTERRA SERIES ALGORITHM

The Volterra series model is the most widely used model for nonlinear systems for several reasons. In particular, this model is useful for nonlinear adaptive filtering because the classical formulation of linear adaptive filters can be easily extended to fit this model. The Volterra series expansion of a nonlinear system consists of a nonrecursive series in which the output signal is related to the input signal as follows[2]

$$\begin{aligned}
d'(k) &= \sum_{l_1=0}^{\infty} w_{o1}(l_1)x(k-l_1) \\
&+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty} w_{o2}(l_1,l_2)x(k-l_1)x(k-l_2) \\
&+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty}\sum_{l_3=0}^{\infty} w_{o3}(l_1,l_2,l_3)x(k-l_1)x(k-l_2)x(k-l_3) \\
&+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty}\cdots \\
&\sum_{l_i=0}^{\infty} w_{oi}(l_1,l_2,\ldots,l_i)x(k-l_1)x(k-l_2)\cdots x(k-l_i) \\
&+\cdots \qquad (11.1)
\end{aligned}$$

where $w_{oi}(l_1,l_2,\ldots,l_i)$, for $i=0,1,\ldots,\infty$, are the coefficients of the nonlinear filter model based on the Volterra series, and $d'(k)$ represents, in the context of system identification application, the unknown system output when no measurement noise exists. The term $w_{oi}(l_1,l_2,\ldots,l_i)$ is also known as the Volterra kernel of the system. Note that the input signals in this case are assumed to consist of a tapped-delay line. For the general case, where the signals of the input signal vector come from different origins, such as in an antenna array, the Volterra series representation is given by

$$\begin{aligned}
d'(k) &= \sum_{l_1=0}^{\infty} w_{o1}(l_1)x_{l_1}(k) \\
&+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty} w_{o2}(l_1,l_2)x_{l_1}(k)x_{l_2}(k) \\
&+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty}\sum_{l_3=0}^{\infty} w_{o3}(l_1,l_2,l_3)x_{l_1}(k)x_{l_2}(k)x_{l_3}(k) \\
&+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty}\cdots \\
&\sum_{l_i=0}^{\infty} w_{oi}(l_1,l_2,\ldots,l_i)x_{l_1}(k)x_{l_2}(k)\cdots x_{l_i}(k) \\
&+\cdots \qquad (11.2)
\end{aligned}$$

[2]The reader should note that the Volterra series expansion includes a constant term w_{o0} which is irrelevant for our discussions here, and will not be further included in the expansion.

where $w_{oi}(l_1, l_2, \ldots, l_i)$, for $i = 0, 1, \ldots, \infty$, are the coefficients of the nonlinear combiner model based on the Volterra series.

As discussed by Mathews [1], the Volterra series expansion can be interpreted as a Taylor series expansion with memory. As such, the Volterra series representation is not suitable to model systems containing discontinuities on their models, as occurs with the Taylor series representation of functions with discontinuities. Another clear drawback of the Volterra series representation is the computational complexity, if the complete series is employed. By truncating the series one can reduce the computational complexity by sacrificing the accuracy of the series expansion. With reduced order, the Volterra series representation is quite complex even when the orders of the series and the filter are moderate. The interested reader can also refer to [2] for a deeper treatment of fixed and adaptive polynomial signal processing.

11.2.1 LMS Volterra Filter

In this subsection, the Volterra LMS algorithm is presented for a second-order series and Nth-order filter. This choice reduces the computational complexity to an acceptable level for some applications and also simplifies the derivations. The extension for higher-order cases is straightforward. The adaptive filter that estimates the signal $d'(k)$ using a truncated Volterra series expansion of second order, can be described by

$$y(k) = \sum_{l_1=0}^{N} w_{l_1}(k)x(k-l_1) + \sum_{l_1=0}^{N}\sum_{l_2=0}^{N} w_{l_1,l_2}(k)x(k-l_1)x(k-l_2) \tag{11.3}$$

where $w_{l_1}(k)$ and $w_{l_1,l_2}(k)$, for $l_1, l_2 = 0, 1, \ldots, N$, are the coefficients of the nonlinear filter model based on the second-order Volterra series expansion, and $y(k)$ represents the adaptive-filter output signal.

The standard approach to derive the LMS algorithm is to use as estimate of the mean-square error (MSE) defined as

$$F[e(k)] = \xi(k) = E[e^2(k)] = E[d^2(k) - 2d(k)y(k) + y^2(k)] \tag{11.4}$$

the instantaneous square error given by

$$e^2(k) = d^2(k) - 2d(k)y(k) + y^2(k) \tag{11.5}$$

Most of the analyses and algorithms presented for the linear LMS apply equally to the nonlinear LMS filter case, if we interpret the information and coefficient vectors as follows

$$\mathbf{x}(k) = \begin{bmatrix} x(k) \\ x(k-1) \\ \vdots \\ x(k-N) \\ x^2(k) \\ x(k)x(k-1) \\ \vdots \\ x(k)x(k-N) \\ \vdots \\ x(k-N)x(k-N+1) \\ x^2(k-N) \end{bmatrix} \tag{11.6}$$

$$\mathbf{w}(k) = \begin{bmatrix} w_0(k) \\ w_1(k) \\ \vdots \\ w_N(k) \\ w_{0,0}(k) \\ w_{0,1}(k) \\ \vdots \\ w_{0,N}(k) \\ \vdots \\ w_{N,N-1}(k) \\ w_{N,N}(k) \end{bmatrix} \tag{11.7}$$

As illustrated in Fig. 11.2, the adaptive-filter output is given by

$$y(k) = \mathbf{w}^T(k)\mathbf{x}(k) \tag{11.8}$$

The estimate of the MSE objective function can now be rewritten as

$$e^2(k) = d^2(k) - 2d(k)\mathbf{w}^T(k)\mathbf{x}(k) + \mathbf{w}^T(k)\mathbf{x}(k)\mathbf{x}^T(k)\mathbf{w}(k) \tag{11.9}$$

An LMS-based algorithm can be used to minimize the objective function as follows:

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) - \mu\hat{\mathbf{g}}_{\mathbf{w}}(k) \\ &= \mathbf{w}(k) - 2\mu e(k)\frac{\partial e(k)}{\partial \mathbf{w}(k)} \end{aligned} \tag{11.10}$$

for $k = 0, 1, 2, \ldots$, where $\hat{\mathbf{g}}_{\mathbf{w}}(k)$ represents an estimate of the gradient vector of the objective function with respect to the filter coefficients. However, it is wise to have different convergence factors for the first- and second-order terms of the LMS Volterra filter. In this case, the updating equations are

Algorithm 11.1

Volterra LMS Algorithm

```
Initialization
  $\mathbf{x}(0) = \mathbf{w}(0) = [0\,0 \ldots 0]^T$
Do for $k \geq 0$
  $e(k) = d(k) - \mathbf{x}^T(k)\mathbf{w}(k)$
  $\mathbf{w}(k+1) = \mathbf{w}(k) + 2 \begin{bmatrix} \mu_1 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & \cdots & \mu_1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \mu_2 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & \mu_2 \end{bmatrix} e(k)\mathbf{x}(k)$
```

given by

$$w_{l_1}(k+1) = w_{l_1}(k) + 2\mu_1 e(k)x(k-l_1) \tag{11.11}$$
$$w_{l_1,l_2}(k+1) = w_{l_2}(k) + 2\mu_2 e(k)x(k-l_1)x(k-l_2) \tag{11.12}$$

where $l_1 = 0, 1, \ldots, N$ and $l_2 = 0, 1, \ldots, N$. As can be observed in Algorithm 11.1, the Volterra LMS algorithm has the same form as the conventional LMS algorithm except for the form of the input vector $\mathbf{x}(k)$. In order to guarantee convergence of the coefficients in the mean, the convergence factor of the Volterra LMS algorithm must be chosen in the range

$$0 < \mu_1 < \frac{1}{tr(\mathbf{R})} < \frac{1}{\lambda_{\max}} \tag{11.13}$$
$$0 < \mu_2 < \frac{1}{tr(\mathbf{R})} < \frac{1}{\lambda_{\max}} \tag{11.14}$$

where $\lambda_{\max}$ is the largest eigenvalue of the input signal vector autocorrelation matrix $\mathbf{R} = E[\mathbf{x}(k)\mathbf{x}^T(k)]$. It should be noted that this matrix involves high-order statistics of the input signal, leading to high eigenvalue spread of the matrix $\mathbf{R}$ even if the input signal is a white noise. As a consequence, the Volterra LMS algorithm has in general slow convergence. As an alternative, we can consider implementing a Volterra adaptive filter using an RLS algorithm.

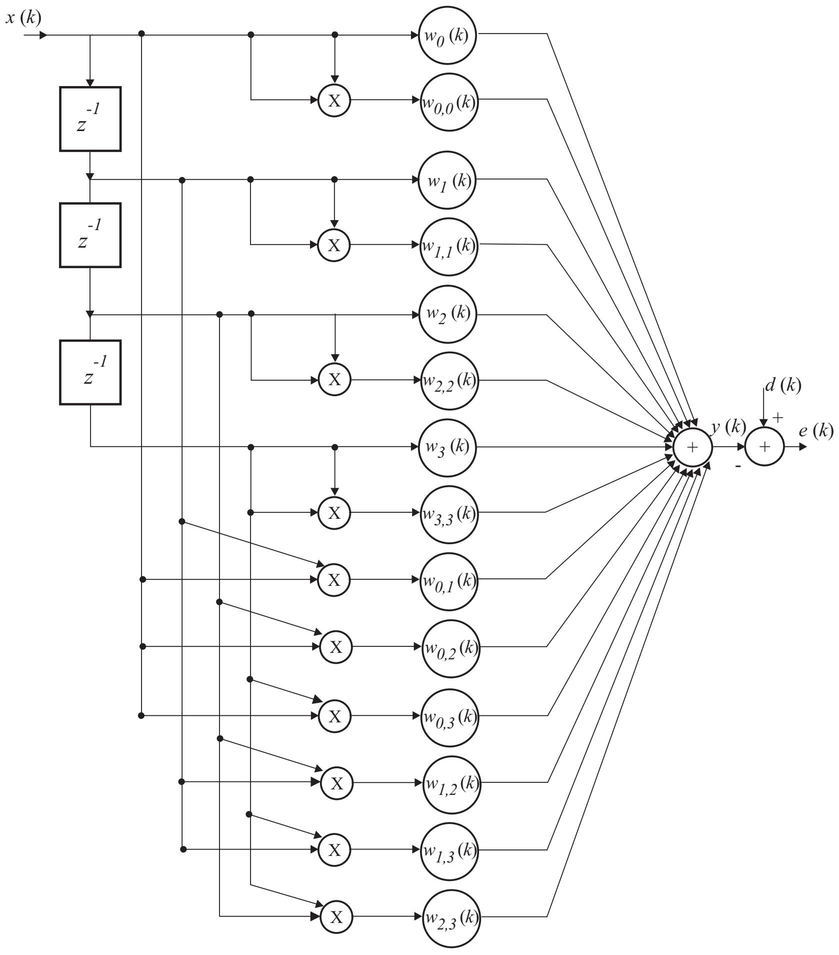


Figure 11.2 Adaptive Volterra filter.

11.2.2 RLS Volterra Filter

The RLS algorithms are known to achieve fast convergence even when the eigenvalue spread of the input vector correlation matrix is large. The objective of the RLS algorithm is to choose the coefficients of the adaptive filter such that the output signal $y(k)$, during the period of observation, will match the desired signal as closely as possible in the least-squares sense. This minimization process can be easily adapted to the nonlinear adaptive filtering case by reinterpreting the entries of the input signal vector and the coefficient vector, as done in the LMS case.

In the case of the RLS algorithm, the deterministic objective function is given by

$$\begin{aligned}\xi^d(k) &= \sum_{i=0}^{k} \lambda^{k-i} \varepsilon^2(i) \\ &= \sum_{i=0}^{k} \lambda^{k-i} \left[d(i) - \mathbf{x}^T(i)\mathbf{w}(k)\right]^2 \end{aligned} \tag{11.15}$$

where $\varepsilon(i)$ is the output error at instant i and

$$\mathbf{x}(i) = \begin{bmatrix} x(i) \\ x(i-1) \\ \vdots \\ x(i-N) \\ x^2(i) \\ x(i)x(i-1) \\ \vdots \\ x(i)x(i-N) \\ \vdots \\ x(i-N)x(i-N+1) \\ x^2(i-N) \end{bmatrix} \tag{11.16}$$

$$\mathbf{w}(k) = \begin{bmatrix} w_0(k) \\ w_1(k) \\ \vdots \\ w_N(k) \\ w_{0,0}(k) \\ w_{0,1}(k) \\ \vdots \\ w_{0,N}(k) \\ \vdots \\ w_{N,N-1}(k) \\ w_{N,N}(k) \end{bmatrix} \tag{11.17}$$

are the input and the adaptive-filter coefficient vectors, respectively. The parameter λ is an exponential weighting factor that should be chosen in the range $0 \ll \lambda \leq 1$.

By differentiating $\xi^d(k)$ with respect to $\mathbf{w}(k)$ and equating the result to zero, the optimal vector $\mathbf{w}(k)$ that minimizes the least-squares error can be shown to be given by

$$\begin{aligned}\mathbf{w}(k) &= \left[\sum_{i=0}^{k} \lambda^{k-i}\mathbf{x}(i)\mathbf{x}^T(i)\right]^{-1} \sum_{i=0}^{k} \lambda^{k-i}\mathbf{x}(i)d(i) \\ &= \mathbf{R}_D^{-1}(k)\mathbf{p}_D(k) \end{aligned} \tag{11.18}$$

where $\mathbf{R}_D(k)$ and $\mathbf{p}_D(k)$ are called the deterministic correlation matrix of the input vector and the deterministic cross-correlation vector between the input vector and the desired signal, respectively.

The Volterra RLS algorithm has the same form as the conventional RLS algorithm as shown in Algorithm 11.2, where the only difference is the form of the input vector $\mathbf{x}(k)$.

Algorithm 11.2

Volterra RLS Algorithm

```
Initialization
  $\mathbf{S}_D(-1) = \delta\mathbf{I}$
where $\delta$ can be the inverse of an estimate of the input signal power
  $\mathbf{x}(-1) = \mathbf{w}(-1) = [0\,0\ldots 0]^T$
Do for $k \geq 0$
  $e(k) = d(k) - \mathbf{x}^T(k)\mathbf{w}(k-1)$
  $\boldsymbol{\psi}(k) = \mathbf{S}_D(k-1)\mathbf{x}(k)$
  $\mathbf{S}_D(k) = \frac{1}{\lambda}[\mathbf{S}_D(k-1) - \frac{\boldsymbol{\psi}(k)\boldsymbol{\psi}^T(k)}{\lambda+\boldsymbol{\psi}^T(k)\mathbf{x}(k)}]$
  $\mathbf{w}(k) = \mathbf{w}(k-1) + e(k)\mathbf{S}_D(k)\mathbf{x}(k)$
If necessary compute
  $y(k) = \mathbf{w}^T(k)\mathbf{x}(k)$
  $\varepsilon(k) = d(k) - y(k)$
```

A clear disadvantage of the Volterra RLS algorithm is the high computational complexity which requires an order of N^4 multiplications per output sample. However, by examining closely the form of the input data vector it is possible to conclude that the nonlinear filtering problem can be recast into a linear multichannel adaptive-filtering problem for which fast RLS algorithms exist. Using this strategy, several fast RLS algorithms for Volterra filters have been proposed, namely the fast transversal [3], the lattice and QR-based lattice algorithms [4], and the QR-decomposition-based algorithm [5]. Other strategies to reduce computation while trying to retain fast convergence, include the orthogonal lattice-based structures tailored for Gaussian input signals [6].

Example 11.1

A digital channel model can be represented by the following system of equations

$$\begin{aligned} v(k) &= x(k) + 0.5x(k-1) \\ y(k) &= v(k) + 0.2v^2(k) + 0.1v^3(k) + n(k) \end{aligned}$$

The channel is corrupted by Gaussian white noise with variance σ_n^2, varying from -10dB to -25dB. The training signal and the actual input signal, consist of independent binary samples (-1,1). The training period depends on the algorithm but our first attempt is 200 iterations, and after that one can start normal operation.

(a) Design an equalizer for this problem. Use a filter of appropriate order and plot the learning curves.

(b) Using the same number of adaptive-filter coefficients, implement a DFE equalizer as shown in Fig. 11.3 and compare the results with those obtained with the FIR equalizer.

We start with the normalized LMS and after making it work, we compare it with the:

1. DFE normalized LMS algorithm
2. Volterra normalized LMS algorithm
3. DFE Volterra normalized LMS algorithm

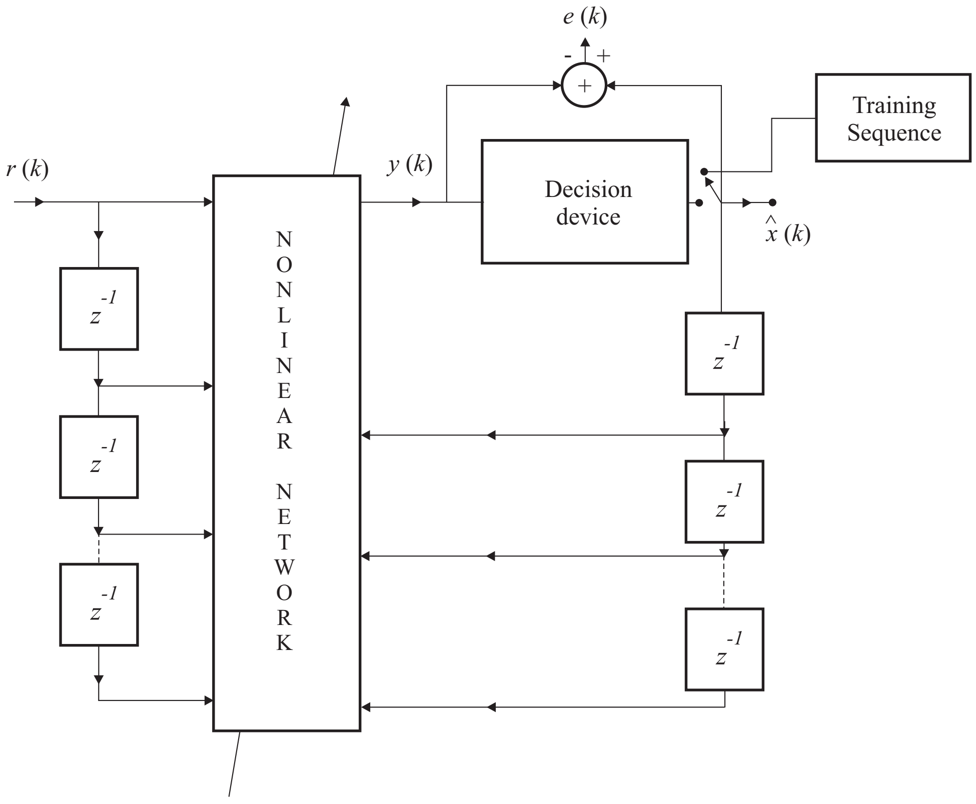


Figure 11.3 Decision feedback equalizer.

Solution:

In the DFE of Fig. 11.3, we initially utilize a training sequence which consists of a properly delayed version of the transmitted signal which is known to the receiver. Obviously, this is an overhead to the communication system since in the beginning no information is actually being transmitted. After the training period no actual reference signal is available, and the equalizer replaces the training sequence by the output of the decision device by moving the switch to its output. The average of square error to be presented corresponds to average of a hundred experiments, whereas the number of errors are measured in single run experiments.

For the normalized LMS algorithm the number of coefficients is 10 with convergence factor $\mu = 0.2$. The square errors for the different levels of channel noise are depicted in Fig. 11.4. As can be observed, the normalized LMS algorithm converges fast for this example where only few training samples are required to train the filter, when the signal to noise ratio is high. However, since the channel is nonlinear the square error after convergence does not reach low levels.

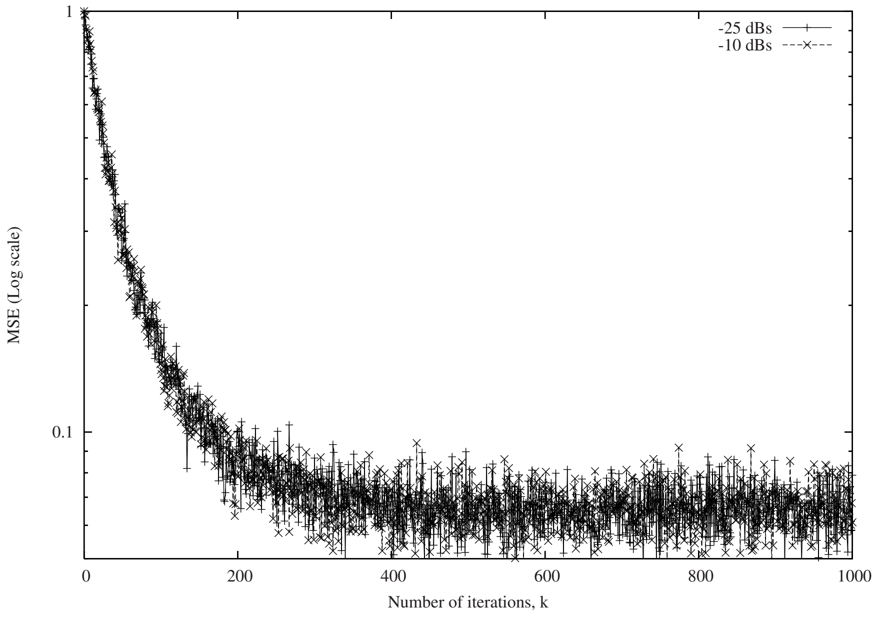


Figure 11.4 Square error, normalized LMS algorithm.

In the next experiment, the decision feedback equalizer is tested using the normalized LMS algorithm with convergence factors $\mu = 0.2$ for the forward and feedback adaptive filters. The forward filter has eight coefficients whereas the feedback filter has two coefficients. For comparison, the results presented are the same as in the previous case for the same levels of channel noise. The resulting square errors are depicted in Fig. 11.5. In this case, the algorithm requires a somewhat comparable training period and also leads to similar square error after convergence. When the signal to noise ratio is poor the standard and the DFE algorithms perform poorly.

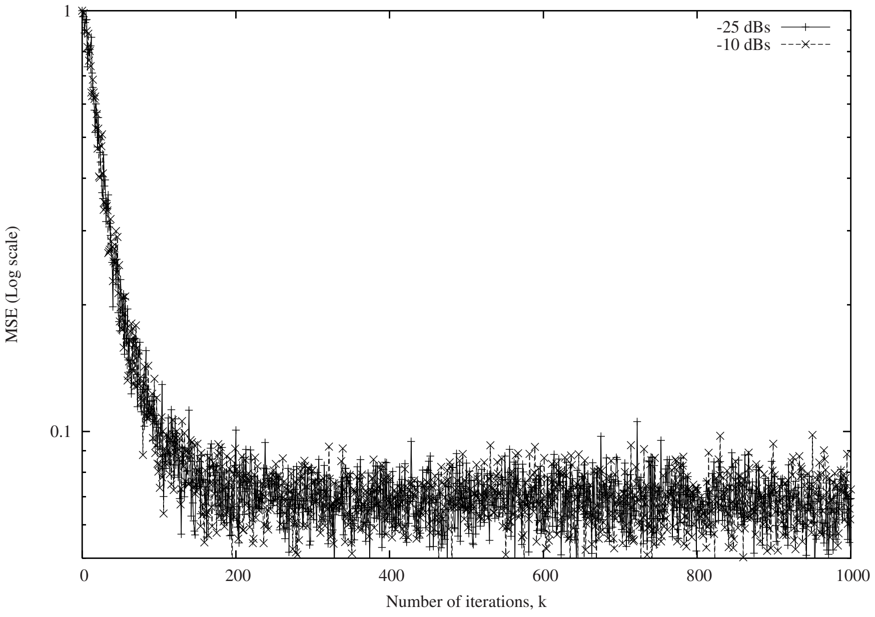


Figure 11.5 Square error for the experiments with the DFE normalized LMS algorithm.

The normalized LMS Volterra series algorithm is also tested in this experiment using a tapped delay line as input with ten elements. The convergence factor for the first-order adaptive coefficients is $\mu_1 = 0.51$ and for the second-order coefficients is $\mu_2 = 0.08$. The results are depicted in Fig. 11.6. A distinct feature of the Volterra algorithm is its lower square error after convergence, which is a consequence of the fact that it models the channel better. Its training period is usually longer due to the larger number of coefficients and higher conditioning number of the information matrix.

We also test the Volterra series algorithm on a decision feedback equalizer. In the feedforward filter a tapped-delay line with eight coefficients is used whereas in the feedback filter two taps are employed. For these experiments the convergence factors used in the coefficients multiplying the linear terms of the forward filter are $\mu_1 = 0.15$ and $\mu_2 = 0.08$, respectively. For the feedback adaptive filter the chosen factors are $\mu_1 = 0.2$ and $\mu_2 = 0.08$, respectively. For comparison the results are presented for the same levels of channel noise as the previous examples. These square errors are seen in Fig. 11.7. The comparison between the DFE and non DFE Volterra filter implementation shows that the DFE requires comparable training period while achieving lower square error and requiring less computational effort. As expected, in all examples the lower additional noise leads to lower MSE after convergence.

Table 11.1 illustrates the number of decision errors made in a single run of the algorithms analyzed in this example. The table also contains the iteration number after which no decision errors are noticed. As can be observed the DFE algorithms usually take longer to converge. Also, the Volterra algorithms have longer learning periods.

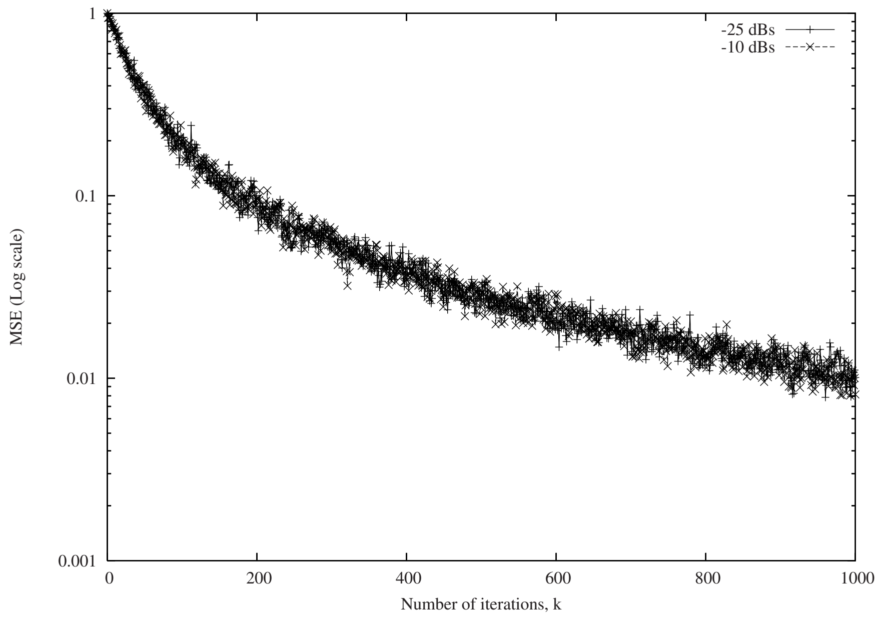


Figure 11.6 Square error for the experiments with the Volterra normalized LMS algorithm.

□

Table 11.1 Evaluation of the Volterra LMS Algorithms

	Noise Level	NLMS	DFE NLMS	Volterra	DFE Volterra
No. of Errors	-25 dBs	2	8	7	9
No. of Errors	-10 dBs	9	11	12	17
Last Error Iter.	-25 dBs	4	30	26	50
Last Error Iter.	-10 dBs	23	25	102	168

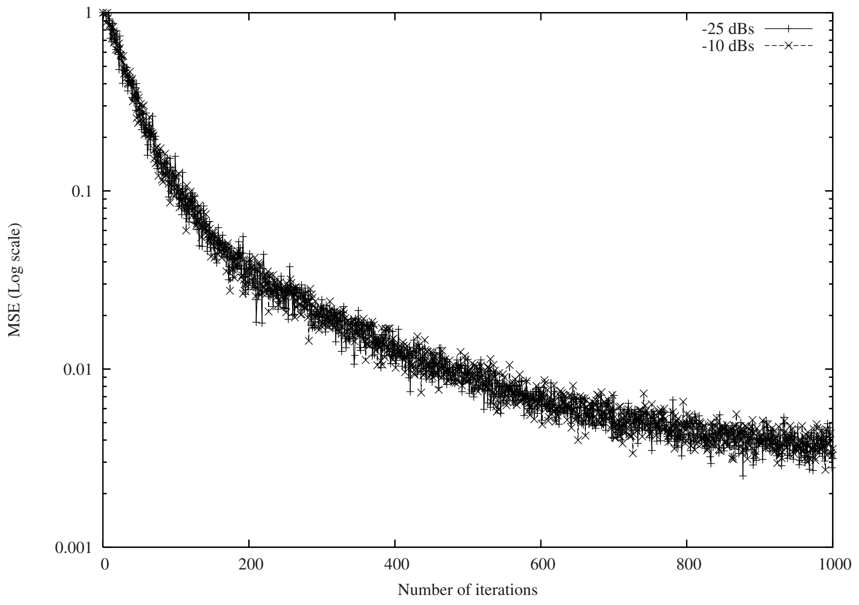


Figure 11.7 Square error for the experiments with DFE Volterra series algorithm.

11.3 ADAPTIVE BILINEAR FILTERS

As it is widely known, the reduction in the computational complexity is the main advantage the adaptive IIR filters present when compared with the adaptive FIR filters. Motivated by this observation, we can consider implementing nonlinear adaptive filters via a nonlinear difference equation, in order to reduce the computational burden related to the Volterra series expansion. The most widely accepted nonlinear difference equation model used for adaptive filtering is the so-called bilinear equation given by

$$y(k) = \sum_{m=0}^{M} b_m(k)x(k-m) - \sum_{j=1}^{N} a_j(k)y(k-j) + \sum_{i=0}^{I}\sum_{l=1}^{L} c_{i,l}x(k-i)y(k-l) \quad (11.19)$$

where $y(k)$ is the adaptive-filter output.

A bilinear adaptive filter in most cases requires fewer coefficients than the Volterra series adaptive filter in order to achieve a given performance. The advantages of the adaptive bilinear filters come with a number of difficulties, some of them not encountered in the Volterra series adaptive filters.

In the present case, the signal information vector is defined by

$$\boldsymbol{\phi}(k) = \begin{bmatrix} x(k) \\ x(k-1) \\ \vdots \\ x(k-M) \\ y(k-1) \\ y(k-2) \\ \vdots \\ y(k-N) \\ x(k)y(k-1) \\ \vdots \\ x(k-I)y(k-L+1) \\ x(k-I)y(k-L) \end{bmatrix} \tag{11.20}$$

where N, M, I and L are the orders of the adaptive-filter difference equations. The coefficient vector can then be described as

$$\boldsymbol{\theta}(k) = \begin{bmatrix} b_0(k) \\ b_1(k) \\ \vdots \\ b_M(k) \\ -a_1(k) \\ -a_2(k) \\ \vdots \\ -a_N(k) \\ c_{0,1}(k) \\ \vdots \\ c_{I,L-1}(k) \\ c_{I,L}(k) \end{bmatrix} \tag{11.21}$$

A possible objective function for adaptive bilinear filtering based on output error is the least-squares function[3]

$$\begin{aligned} \xi^d(k) &= \sum_{i=0}^{k} \lambda^{k-i} e^2(i) \\ &= \sum_{i=0}^{k} \lambda^{k-i} [d(i) - \boldsymbol{\theta}^T(k)\boldsymbol{\phi}(i)]^2 \end{aligned} \tag{11.22}$$

[3]Like in Chapter 10, the reader should note that this definition of the deterministic weighted least squares utilizes the *a priori* error with respect to the latest data pair $d(k)$ and $x(k)$, unlike the FIR RLS case.

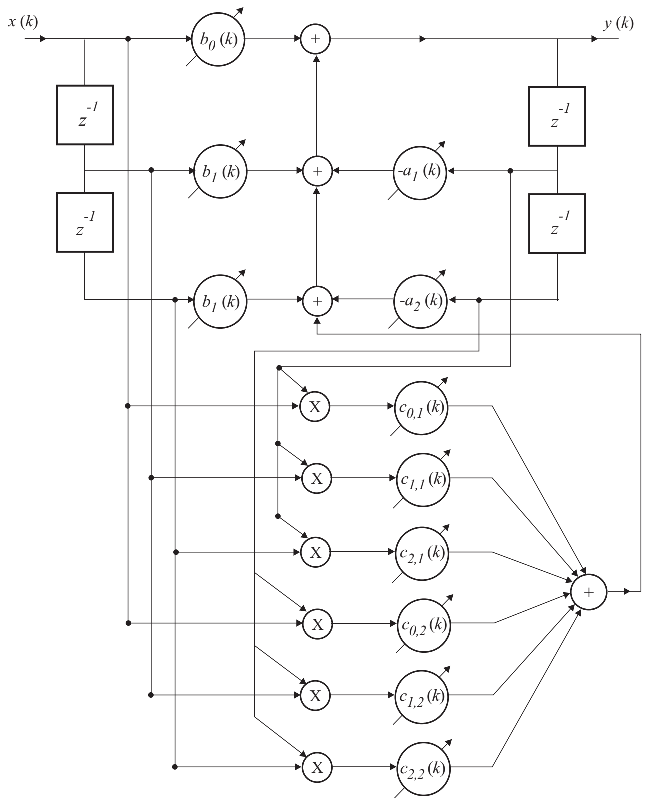


Figure 11.8 Adaptive bilinear filter.

The forgetting factor λ as usual is chosen in the range $0 << \lambda < 1$. By differentiating $\xi^d(k)$ with respect to $\boldsymbol{\theta}(k)$, and by using the same arguments used to deduce the output error RLS algorithm for linear IIR adaptive filters, we conclude that the RLS algorithm for adaptive bilinear filtering consists of the following basic steps:

$$e(k) = d(k) - \boldsymbol{\theta}^T(k)\boldsymbol{\phi}(k) \tag{11.23}$$

$$\boldsymbol{\varphi}(k) = -\frac{\partial y(k)}{\partial \boldsymbol{\theta}(k)} \approx -\boldsymbol{\phi}(k) \tag{11.24}$$

$$\mathbf{S}_D(k+1) = \frac{1}{\lambda}\left[\mathbf{S}_D(k) - \frac{\mathbf{S}_D(k)\boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k)\mathbf{S}_D(k)}{\lambda + \boldsymbol{\varphi}^T(k)\mathbf{S}_D(k)\boldsymbol{\varphi}(k)}\right] \tag{11.25}$$

$$\boldsymbol{\theta}(k+1) = \boldsymbol{\theta}(k) - \mathbf{S}_D(k+1)\boldsymbol{\varphi}(k)e(k) \tag{11.26}$$

The approximation of equation (11.24) is not accurate, however it is computationally simple and simulation results confirm that it works. The reader should notice that the partial derivatives used in this algorithm are only approximations, leading to a suboptimal RLS solution. More accurate approximations can be derived by following the same reasonings in which the partial derivatives were calculated for the output error RLS algorithm for linear IIR adaptive filters. The description of the bilinear RLS algorithm is given in Algorithm 11.3.

Algorithm 11.3

Bilinear RLS Algorithm

```
Initialization
  $a_i(k) = b_i(k) = c_{i,l}(k) = e(k) = 0$
  $y(k) = x(k) = 0\,,\ k\ < 0$
  $\mathbf{S}_D(0) = \delta^{-1}\mathbf{I}$
For each $x(k),\ d(k),\ k \geq 0,$ do
  $y(k) = \boldsymbol{\phi}^T(k)\boldsymbol{\theta}(k)$
  $e(k) = d(k) - y(k)$
  $\mathbf{S}_D(k+1) = \frac{1}{\lambda}\left[\mathbf{S}_D(k) - \frac{\mathbf{S}_D(k)\boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k)\mathbf{S}_D(k)}{\lambda+\boldsymbol{\varphi}^T(k)\mathbf{S}_D(k)\boldsymbol{\varphi}(k)}\right]$
  $\boldsymbol{\theta}(k+1) = \boldsymbol{\theta}(k) - \mathbf{S}_D(k+1)\boldsymbol{\varphi}(k)e(k)$
   Stability test
```

If we consider as objective function the mean-square error (MSE) defined as

$$\xi = E[e^2(k)] \tag{11.27}$$

we can derive a gradient-based algorithm, by using $e^2(k)$ as an estimate for ξ, leading to an updating equation given by

$$\boldsymbol{\theta}(k+1) = \boldsymbol{\theta}(k) - 2\begin{bmatrix} \mu_1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & \cdots & \mu_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \mu_2 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & \mu_2 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \mu_3 & \cdots & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & \mu_3 \end{bmatrix} \boldsymbol{\varphi}(k)e(k) \tag{11.28}$$

where

$$e(k) = d(k) - \boldsymbol{\theta}^T(k)\boldsymbol{\phi}(k) \tag{11.29}$$

and

$$\boldsymbol{\varphi}(k) = \frac{\partial e(k)}{\partial \boldsymbol{\theta}(k)} \tag{11.30}$$

Again, the calculation of an accurate gradient vector can be quite cumbersome.

The main drawbacks of the adaptive bilinear filters based on the output error are: possible instability of the adaptive filter [25], slow convergence, and convergence to local minima of the error surface. It is also possible in the case of adaptive bilinear filter to apply an equation error formulation. In the presence of additional noise, the equation error algorithm may also lead to instability or to a biased global minimum solution.

Example 11.2

Identify an unknown system with the following model

$$d(k) = -0.3d(k-1) + x(k) + 0.04x^2(k) + 0.1x^3(k) + n(k)$$

using the bilinear algorithm, and compare the results with those obtained with the Volterra normalized LMS algorithm. The additional noise is Gaussian white noise with variance $\sigma_n^2 = -10$dB. Use Gaussian white noise with unit variance as input.

Solution:

Three coefficients are sufficient for the bilinear algorithm to perform well. The chosen convergence factor is $\mu = 0.005$. For the Volterra normalized LMS algorithm we use six coefficients and $\mu = 0.1$. As can be observed in Fig. 11.9, the bilinear algorithm converges faster and leads to a lower square error after convergence than the Volterra normalized LMS algorithm, since the unknown system has a bilinear model.

□

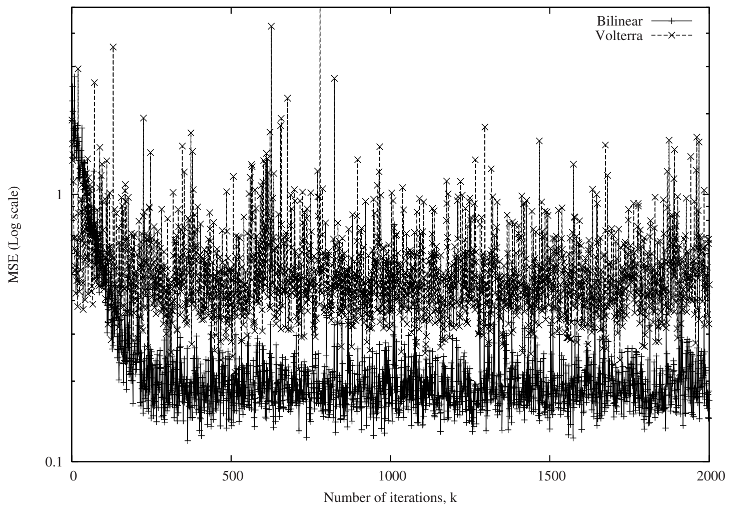


Figure 11.9 Square error for the experiment with the bilinear and Volterra normalized LMS algorithms.

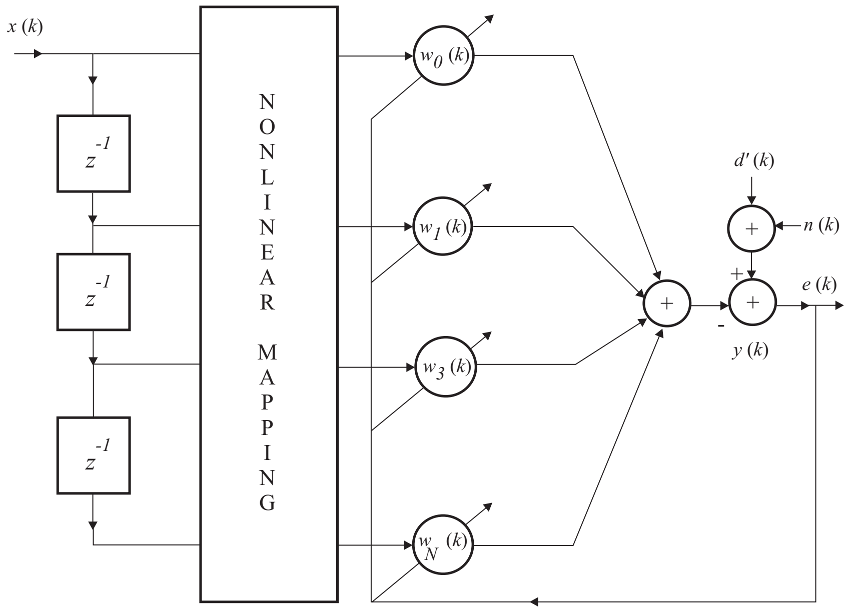


Figure 11.10 Neural network based adaptive filter.

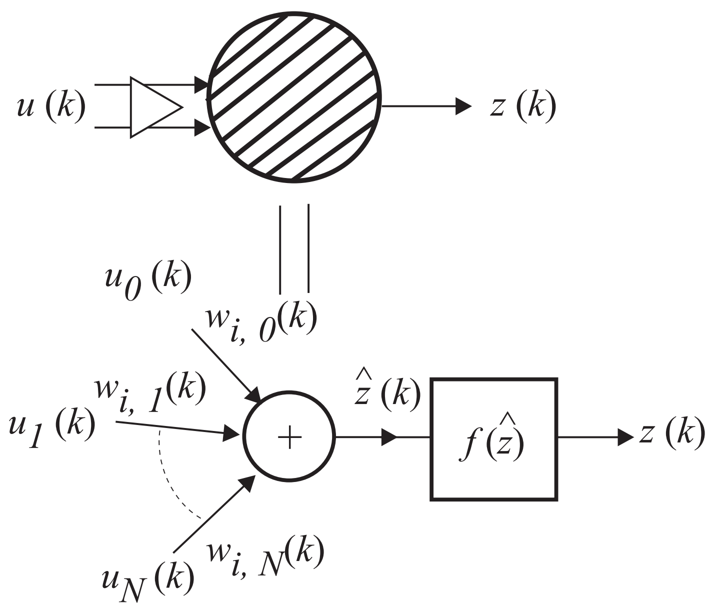
11.4 MULTILAYER PERCEPTRON ALGORITHM

In this section, the multilayer perceptron algorithm is briefly presented [24]. This algorithm belongs to a class of nonlinear adaptive filters where the input signal vector is mapped into another signal vector through a multiport network containing several local nonlinearities, as depicted in Fig. 11.10. Usually, the nonlinear multiport network consists of feedforward neural networks with several layers, where the nonlinearities (neurons) are placed inside the network in a structurally modular form.

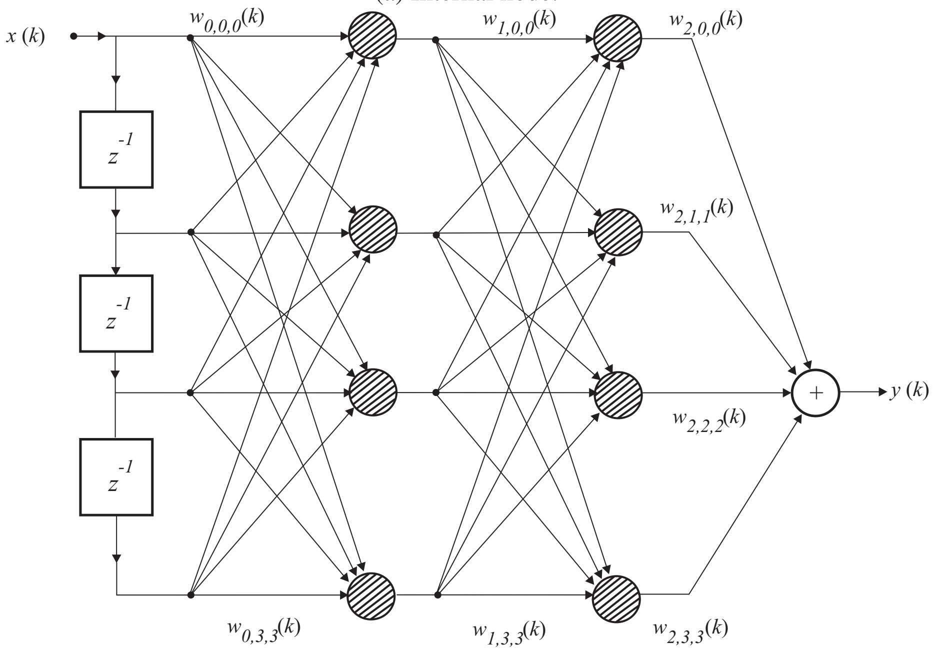
The multilayer perceptron structure consists of several layers including an input layer, an output layer and several internal layers usually called hidden layers. Fig. 11.11 illustrates a multilayer perceptron-based adaptive filter with three layers. In communication applications the output layer usually has a single neuron, with $y(k)$ representing the nonlinear adaptive-filter output signal. The mathematical description for each neuron is

$$y_{l,i}(k) = f_{l,i}\left\{\sum_{j=0}^{N_{l-1}-1} w_{l,i,j}(k)y_{l-1,j}(k) - bs_{l,i}(k)\right\} \tag{11.31}$$

where $w_{l,i,j}(k)$ are the weight coefficients connecting the output signal $y_{l-1,j}(k)$ of the jth neuron from layer $l-1$ to input of neuron i of layer l, for $l = 0, 1, \ldots, L-1$; $i = 0, \ldots, N_l - 1$. Note that N_l is the number of neurons in the lth layer and the index L is the number of layers. Each constant $bs_{l,i}(k)$ is the bias term of the ith neuron at layer l, that is also known as the threshold. It is a well known result that the multilayer perceptron network is able to implement any desired nonlinear mapping by properly choosing the weights, the thresholds and the nonlinear activation function $f\{\cdot\}$



(a) Internal node.



(b) General structure.

Figure 11.11 Multilayer perceptron adaptive filter.

[26]. Although, the activation function and the threshold could be chosen to be different for each layer, we will not consider this general case here. Also, it is possible to show that three layers is always enough for practical purposes. However, the use of more than three layers is desirable in many applications, where in the three layers case the hidden layer requires a large number of neurons in order to achieve an acceptable nonlinear mapping.

The most widely used activation function is the sigmoid function, defined as

$$\mathrm{sgm}(x) = \frac{2c_1}{1+\mathrm{e}^{-c_2 x}} - c_1 \tag{11.32}$$

where c_1 and c_2 are suitably chosen constants. The derivative of the sigmoid function is given by

$$\mathrm{sgd}(x) = \frac{c_2}{2c_1}[c_1^2 - \mathrm{sgm}^2(x)] \tag{11.33}$$

A popular updating algorithm for the multilayer perceptron is the so-called backpropagation algorithm. The objective function is to minimize the instantaneous output square error, that is

$$e^2(k) = [d(k) - y(k)]^2 \tag{11.34}$$

In order to minimize the above objective function, the backpropagation algorithm uses a steepest-decent updating, with the gradient calculated from the output layer to the input layer as following presented. The derivation of the backpropagation algorithm falls beyond the scope of this book, the interested reader should consult [26] or [27]. In the output layer the error signal is given by $e(k)$ itself, as a result the coefficient updating for the coefficients of the output layer is given by

$$w_{L-1,i,j}(k+1) = w_{L-1,i,j}(k) + 2\mu_{L-1}e(k)y_{L-1,j}(k) \tag{11.35}$$

where $i = 0, 1, \ldots, N_{L-2}-1$ and $j = 0, 1, \ldots, N_{L-1}-1$. Notice that in our case we are considering a single output multilayer perceptron, therefore $N_{L-1} = 1$. The parameter μ_{L-1} is the convergence factor for the output layer. Also the simplified updating equation above resulted from not using an activation function at the output node. If the activation function is included at the output node the updating equation is given by

$$w_{L-1,i,j}(k+1) = w_{L-1,i,j}(k) + 2\mu_{L-1}e(k)\mathrm{sgd}\left\{\mathrm{sgm}^{-1}[y_{L-1,j}(k)]\right\}\mathrm{sgm}[y_{L-2,j}(k)] \tag{11.36}$$

Since we know the error in the output layer, we can propagate this error backwards, and calculate the corresponding errors in the output of the internal neurons. By examining Fig. 11.11 closely, after applying the chain rule for derivative and performing a number of manipulations (see [26] and [27] for details) it is possible to show that the error signal at the jth neuron from layer l is given by

$$\begin{aligned} e_{l,j}(k) &= \mathrm{sgd}\left\{\mathrm{sgm}^{-1}[y_{l,j}(k)]\right\} \sum_{i=0}^{N_l-1} w_{l+1,i,j}(k)e_{l+1,i}(k) \\ &= \mathrm{sgd}\left[\sum_{j=0}^{N_{l-1}-1} w_{l,i,j}(k)y_{l-1,j}(k)\right] \sum_{i=0}^{N_l-1} w_{l+1,i,j}(k)e_{l+1,i}(k) \end{aligned} \tag{11.37}$$

The updating equations for the coefficients of the internal layers and the bias terms are given by

$$\begin{aligned} w_{l,i,j}(k+1) &= w_{l,i,j}(k) + 2\mu_l e_{l,j}(k) y_{l-1,j}(k) \\ bs_{l,i}(k+1) &= bs_{l,i}(k) - 2\mu_l e_{l,j}(k) \end{aligned} \tag{11.38}$$

for $i = 0, 1, \ldots, N_{l-1} - 1$ and $j = 0, 1, \ldots, N_l - 1$.

The description of the multilayer perceptron algorithm for nonlinear adaptive filtering is given in Algorithm 11.4.

Algorithm 11.4

Multilayer Perceptron Algorithm

```
Initialization
  Choose each $w_{l,i,j}(0)$ randomly
Do for $k \geq 0$
  Choose $y_{-1,j}(k) = x_j(k)$
  Do for $l = 0, \ldots, L-1$
    Do for $i = 0, \ldots, N_l - 1$
      Do for $j = 0, \ldots, N_{l-1} - 1$
        $y_{l,j}(k) = f_{l,j}\{\sum_{i=0}^{N_{l-1}-1} w_{l,j,i}(k) y_{l-1,i}(k) - bs_{l,j}(k)\}$
      End
    End
  End
  $e(k) = d(k) - y_{L-1,0}(k)$
  Do for $l = L-1, \ldots, 0$
    Do for $i = 0, \ldots, N_l - 1$
      Do for $j = 0, \ldots, N_{l-1} - 1$
      If $l = L-1$
        $w_{L-1,i,j}(k+1) = w_{L-1,i,j}(k) + 2\mu_{L-1} e(k) \text{sgd}\left\{\text{sgm}^{-1}[y_{L-1,j}(k)]\right\} \text{sgm}[y_{L-2,j}(k)]$
      Else
        $e_{l,j}(k) = \text{sgd}\left[\sum_{j=0}^{N_{l-1}-1} w_{l,i,j}(k) y_{l-1,j}(k)\right] \sum_{i=0}^{N_l-1} w_{l+1,i,j}(k) e_{l+1,i}(k)$
        $w_{l,i,j}(k+1) = w_{l,i,j}(k) + 2\mu_l e_{l,j}(k) y_{l-1,j}(k)$
        $bs_{l,i}(k+1) = bs_{l,i}(k) - 2\mu_l e_{l,j}(k)$
      End if
    End
  End
End
```

This algorithm has an increased computational complexity as compared with the linear adaptive filters, for a given number of adaptive coefficients. In addition, the convergence speed is likely to be slow, because we are employing a gradient-based algorithm to search an objective function with a nonquadratic surface.

Some attempts to improve the convergence speed have been proposed, see for example [20]. Despite that, nonlinear adaptive filters based on multilayer perceptron require long training periods, and have no methodology to appropriately define the number of layers and the number of neurons, rendering these algorithms difficult to apply in practical problems. However, it is worth it to search for improved nonlinear solutions for the adaptive-filtering problem, because in many communication applications the linear adaptive filter does not yield good enough performance.

Example 11.3

Identify the same system described in example 11.2 using the multilayer perceptron method, and compare the results with those obtained with the Volterra normalized LMS algorithm.

Solution:

In order to identify the same system of example 11.2 with the multilayer perceptron method, we use a network with 3 inputs and 8 neurons in each of the two hidden layers. The chosen convergence factor is μ=0.1. As can be observed in Figs. 11.9 and 11.12, the multilayer perceptron algorithm has worse performance than the bilinear algorithm, but converges slightly faster and reaches a lower square error after convergence than the Volterra normalized LMS algorithm.

□

11.5 RADIAL BASIS FUNCTION ALGORITHM

The radial basis function network is an attractive alternative to the multilayer perceptron for nonlinear adaptive filtering for a number of reasons. As mentioned in [27], the learning process of the radial basis function neural network is the same as finding a surface in the multidimensional space which is a best fit to the training data. In particular, in the case of communication applications this technique is attractive because its learning allows the division of a multidimensional space in appropriate subregions where each received data fits in.

For equalization problems [23], [21], it is well known that the maximum likelihood equalizer using the Viterbi algorithm provides the best solution, with high computational cost. As a compromise, the radial basis function has been proposed as an attractive alternative because of its lower computational complexity and due to its close relationship with Bayesian methods [22]. The Bayesian methods are effective in interference cancellation and channel equalization [9], [11]-[15]. In fact, the Bayesian design leads to the optimal nonlinear adaptive equalizer [8]. In the Bayesian approach, the decision in favor of a symbol is made only if the probability that the referred symbol had caused the current input signal vector exceeds the probability that any other symbol had caused the same input. The optimal decision boundaries are determined by the values of the input signal vector where these

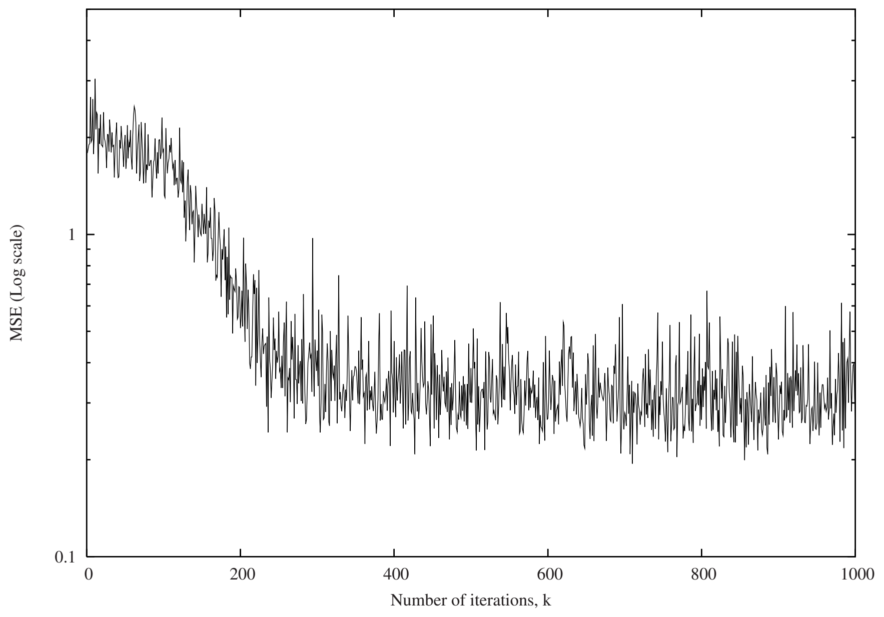


Figure 11.12 Square error for the experiment with the multilayer perceptron algorithm.

probabilities are the same. The Bayesian theory shows that in a number of situations the optimal decision boundaries are not given by hyperplanes (the only ones realizable with linear equalizers), but by nonplanar boundaries. This is exactly what happens when the channel model in communication systems cannot be well modeled with linear adaptive systems, or if the channel transfer function does not have minimum phase. Also, the linear adaptive equalizer does not explore the fact that the input signal originates from transmitted signals consisting of a finite set of symbols.

Since the radial basis function can approximate the Bayesian solution within a reasonable training time, it is a potential candidate to be employed in a number of communication applications where nonlinear adaptive filters are required.

The radial basis function network consists of three layers where the first feeds the second layer directly without any weighting (weights are equal to one), and the output layer is just a linear combiner as depicted in Fig. 11.13.b. The hidden layer implements a nonlinear mapping on the input vector, as represented in Fig. 11.13.a, and consists of two steps. In the first step, the input signal vector is compared with a set of reference vectors $\mathbf{r}_i(k)$, for $i = 0, 1, \ldots, N_N - 1$, where N_N is the number of (hidden) neurons. These vectors are called centers. The comparison between the input signal vector and the centers are usually measured through the Euclidean norm as follows

$$d_i(k) = ||\mathbf{x}(k) - \mathbf{r}_i(k)|| \tag{11.39}$$

These distances are then applied to a nonlinear activation function, which is scalar and radially symmetric. Typical choices are the Gaussian and thin-plate-spline functions, respectively given by

$$\begin{aligned} f(d_i(k)) &= \mathrm{e}^{\frac{-d_i^2(k)}{\sigma_i^2(k)}} \\ f(d_i(k)) &= \frac{d_i^2(k)}{\sigma_i^2(k)}\log[\frac{d_i(k)}{\sigma_i(k)}] \end{aligned} \tag{11.40}$$

The parameter $\sigma_i(k)$ controls the spread of the function, related to the radius of influence of radial basis function $f[d_i(k)]$. The output signal is computed by

$$F[\mathbf{x}(k)] = f_2\left\{\sum_{i=0}^{N_N-1} w_i(k)f[d_i(k)]\right\} \tag{11.41}$$

where $f_2\{\cdot\}$ is the activation function of the output signal. This function is usually of the following form

$$f_2(x) = \frac{1-\mathrm{e}^{-cx}}{1+\mathrm{e}^{-cx}} \tag{11.42}$$

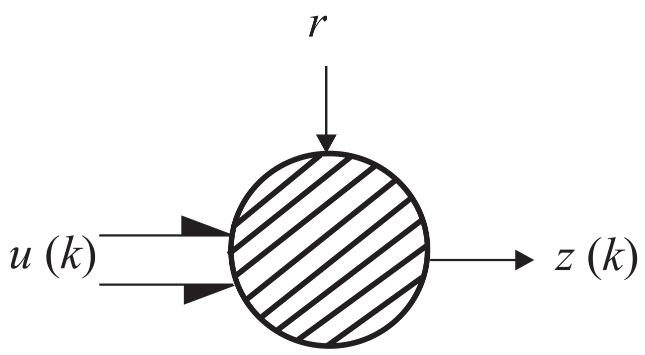
where c is a suitably chosen constant. In most cases, no activation function is used at the output in order to simplify the algorithm, that is $f_2(x) = x$. As a result we will not consider it further.

Usually the training for the radial basis function adaptive filter is done in three steps, where the radius parameters, the centers and the weights are trained separately and in sequence. By using a stochastic gradient algorithm and Gaussian activation function, the radial basis function updating equations are given by

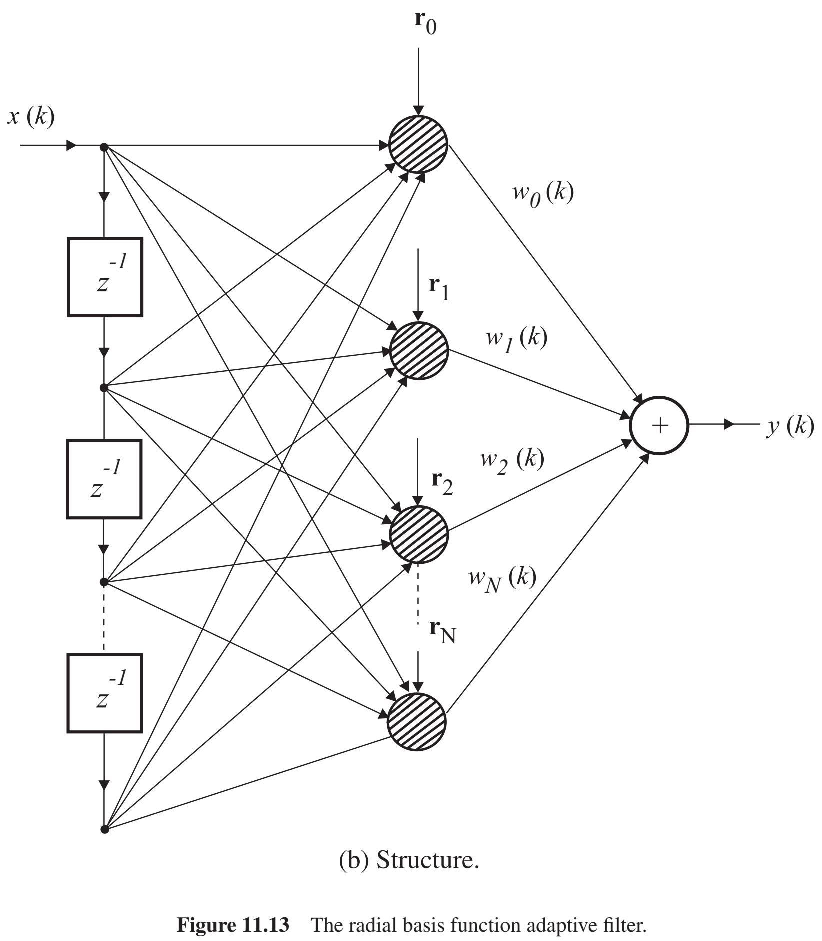
$$\begin{aligned} w_i(k+1) &= w_i(k) + 2\mu_w e(k) f[d_i(k)] \\ \sigma_i(k+1) &= \sigma_i(k) + 2\mu_\sigma e(k) f[d_i(k)] w_i(k)\frac{d_i^2(k)}{\sigma_i^3(k)} \\ \mathbf{r}_i(k+1) &= \mathbf{r}_i(k) + 2\mu_r e(k) f[d_i(k)] w_i(k)\frac{\mathbf{x}(k)-\mathbf{r}_i(k)}{\sigma_i^2(k)} \end{aligned} \tag{11.43}$$

for $i = 0, 1, \ldots, N_N - 1$. In Algorithm 11.5, the adaptive nonlinear filter based on the radial basis function is detailed. In many cases the parameters $\sigma_i(k)$, that control the spread of the function in each neuron, are kept constant, where in this case they are chosen as the expected channel noise power.

In a number of communication applications the signals involved are originally complex. In those cases, we need to use a complex radial basis function algorithm whose configuration is depicted in Fig. 11.14. The complex algorithm is described in Algorithm 11.6, where the derivations are omitted for the sake of brevity, for details consult [16]-[17], [18]-[19].



(a) Internal node.



(b) Structure.

Figure 11.13 The radial basis function adaptive filter.

Algorithm 11.5

Radial Basis Function Algorithm

```
Initialization
  Choose each $w_i(0)$ randomly
Do for $k \geq 0$
  $y(k) = F(\mathbf{x}(k)) = \sum_{i=0}^{N_N-1} w_i(k) f[d_i(k)]$
  $e(k) = d(k) - y(k)$
  Do for $i = 0, 1, \ldots, N_N - 1$
    $w_i(k+1) = w_i(k) + 2\mu_w e(k) f[d_i(k)]$
    $\sigma_i(k+1) = \sigma_i(k) + 2\mu_\sigma f[d_i(k)] e(k) w_i(k) \frac{d_i^2(k)}{\sigma_i^3(k)}$
    $\mathbf{r}_i(k+1) = \mathbf{r}_i(k) + 2\mu_r f[d_i(k)] e(k) w_i(k) \frac{\mathbf{x}(k)-\mathbf{r}_i(k)}{\sigma_i^2(k)}$
  End
End
```

Algorithm 11.6

Complex Radial Basis Function Algorithm

```
Initialization
  Choose each $w_i(0)$ randomly
  $\mathbf{r}_i(k)$, $\mathbf{x}_i(k)$ are complex vectors
  $e(k)$, is a complex scalar
Do for $k \geq 0$
  $y(k) = F(\mathbf{x}(k)) = \sum_{i=0}^{N_N-1} w_i^*(k) f(d_i(k))$
  $e(k) = d(k) - y(k)$
  Do for $i = 0, 1, \ldots, N_N - 1$
    $w_i(k+1) = w_i(k) + 2\mu_w e(k) f[d_i(k)]$
    $\sigma_i(k+1) = \sigma_i(k) + 2\mu_\sigma f[d_i(k)] \{\mathrm{re}[e(k)] w_{R_i}(k) + \mathrm{im}[e(k)] w_{I_i}(k)\} \frac{d_i^2(k)}{\sigma_i^3(k)}$
    $\mathbf{r}_i(k+1) = \mathbf{r}_i(k) + 2\mu_r f[d_i(k)] \frac{\mathrm{re}[e(k)] w_{R_i}(k) \mathrm{re}[\mathbf{x}(k)-\mathbf{r}_i(k)] + \jmath \mathrm{im}[e(k)] w_{I_i}(k) \mathrm{im}[\mathbf{x}(k)-\mathbf{r}_i(k)]}{\sigma_i^2(k)}$
  End
End
```

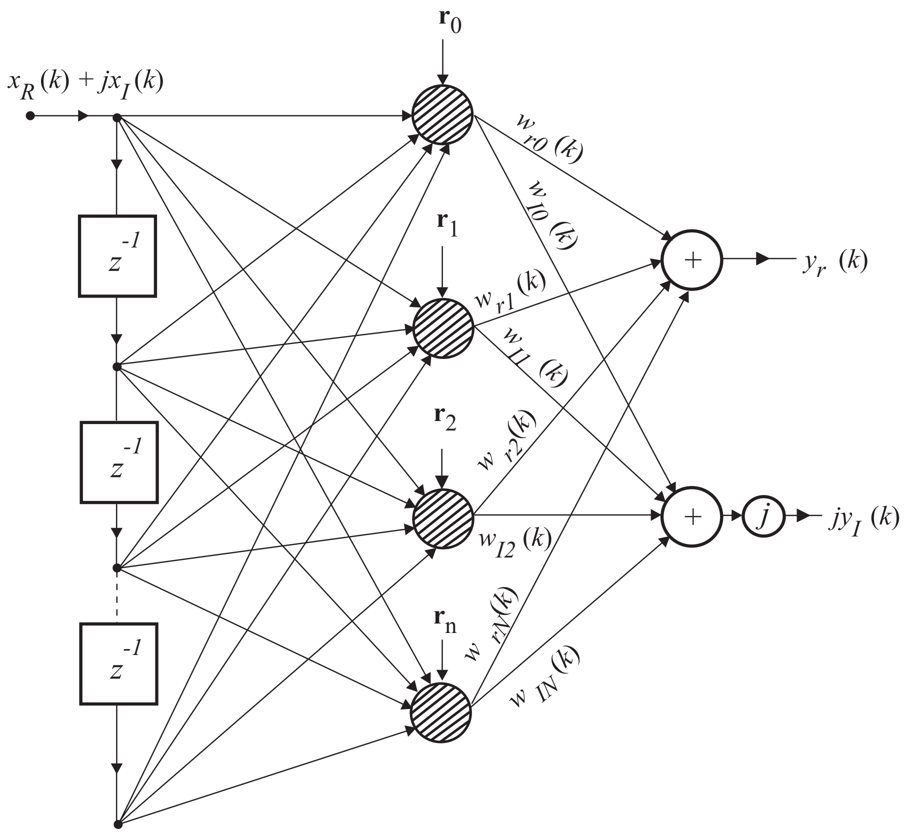


Figure 11.14 The radial basis function adaptive filter for complex signals.

Example 11.4

Solve the problem described in Example 11.1 using:

1. Radial basis function algorithm
2. DFE radial basis function algorithm

Solution:

In order to solve the problem, the following two experiments use neural network equalizers of the radial basis function type with ten delays in the input tap-delay line and ten hidden neurons. In the first experiments the standard radial basis approach is applied using a convergence factor for the linear combiner of $\mu_w = 0.1$, a convergence factor for the radius of $\mu_r = 0.9$, and a spread factor of $\sigma = 0.8$. Fig. 11.15 shows the learning curves for the square errors. As can be observed, the radial basis algorithm requires longer training period than the previous algorithms. This is the price paid by its generality in approximating nonlinear functions.

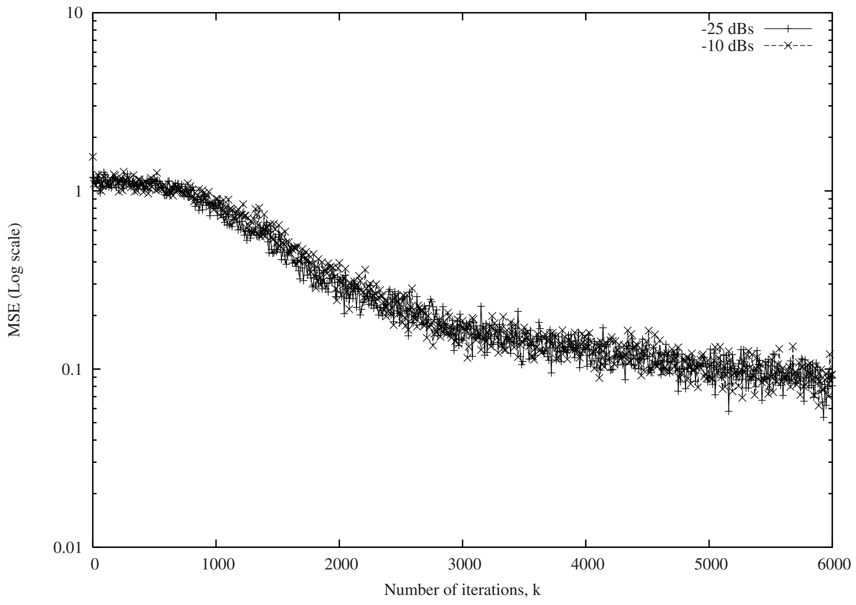


Figure 11.15 Square errors for the experiments with the radial basis algorithm.

The final experiment uses a neural network DFE of the radial basis function type with eight taps and hidden neurons in the forward filter and two in the feedback filter. The convergence factor for the forward filter is $\mu_w = 0.5$, the convergence factor for the radius is $\mu_r = 0.9$, and the spread factor is $\sigma = 0.8$. For the backward filter, these parameters are $\mu_w = 0.04$, $\mu_r = 0.9$, and $\sigma = 1.2$, respectively. These results are depicted in Fig. 11.16 for an ensemble of a hundred experiments. The results with DFE are better than in the case without DFE.

Table 11.2 illustrates the number of decision errors made in a single run of the radial basis function algorithms for this example, including the iteration number after which no decision errors are noticed. As can be observed, the radial basis function algorithms take longer to converge than the Volterra algorithms for this example.

Table 11.2 Evaluation of the Radial Basis Function Algorithms

	Radial Basis Algorithm		**DFE Radial Basis Algorithm**	
Noise level	**-25 dBs**	**-10 dBs**	**-25 dBs**	**-10 dBs**
No. of Errors	74	113	79	92
Iter. of Last Error	318	387	287	370

Fig. 11.17 depicts the results of an experiment with the radial basis function algorithm with DFE where the training is done for a long period. The graphs show that after the learning is complete the

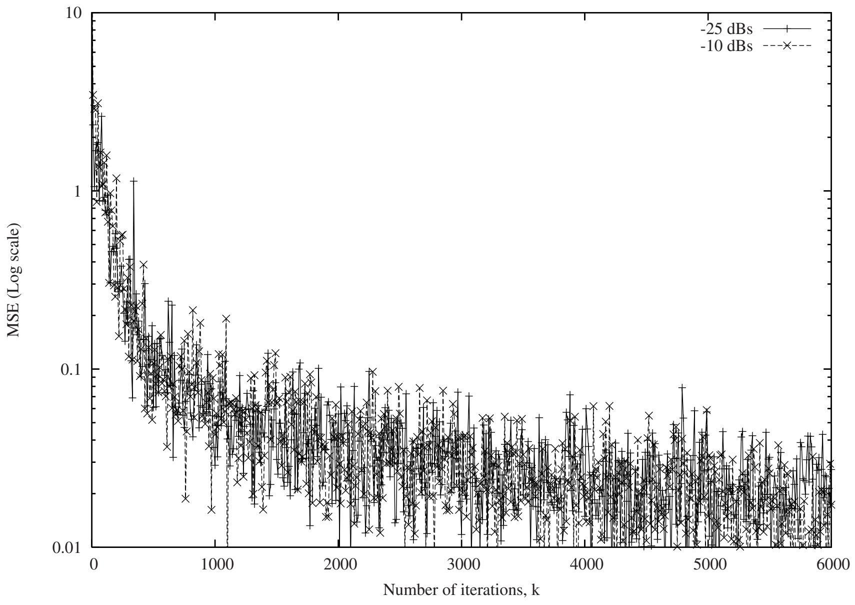


Figure 11.16 Square errors for the experiments with DFE radial basis function algorithm.

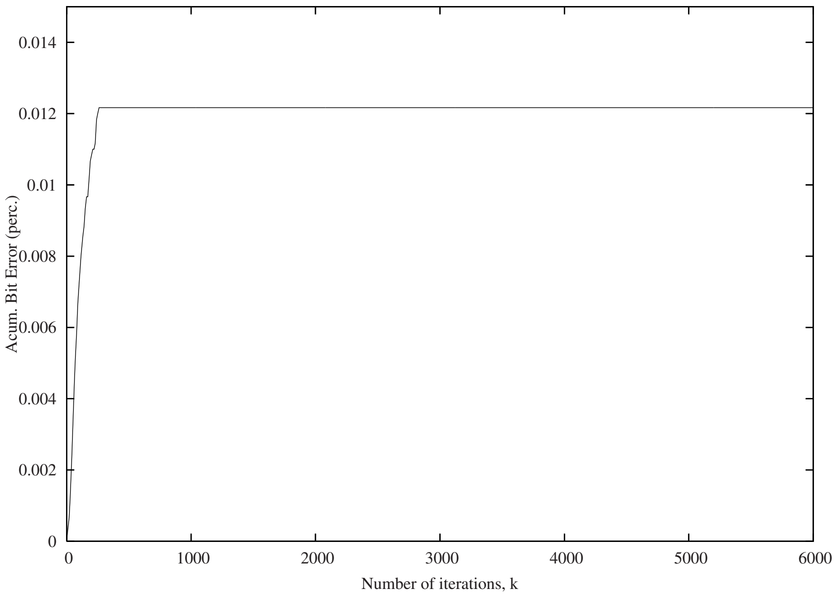
algorithm enables perfect bit detection, reaching a lower square error level than the algorithms not based on neural networks.

□

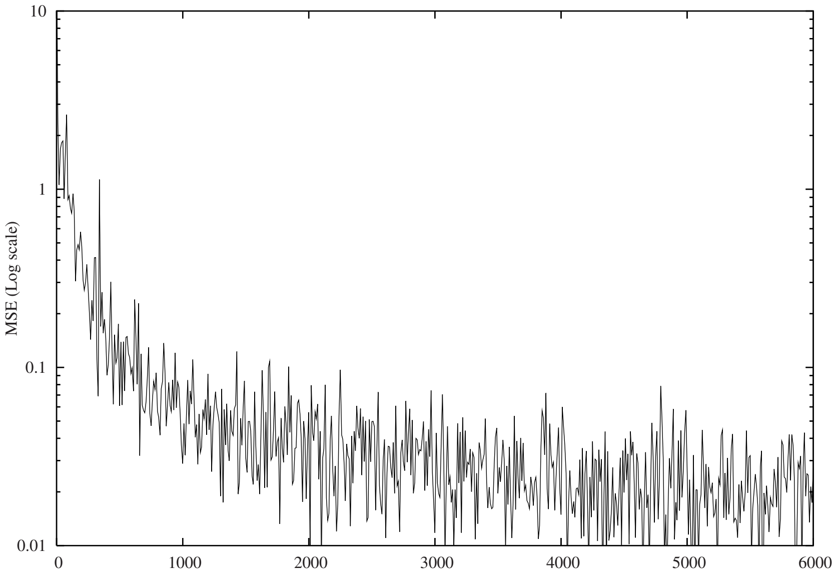
11.6 CONCLUSION

In this chapter, we introduced some nonlinear adaptive-filtering methods which can be applied in communication systems, as well as in many other fields. The methods discussed here are far from consisting of a complete set, many other methods have been investigated using different points of view, see for example [28] and [29]. The emphasis was to describe methods allowing a training period and suitable for channel equalization and co-channel interference. No attempt was made to discuss blind equalization methods that are nonlinear adaptive filters which usually utilize high-order statistics, see Chapter 13.

The wide use of these algorithms in modern communication systems, while required, remains to be seen. However, with a deep knowledge of the type of nonlinearities affecting the given communication environment, one can come up with a nonlinear adaptive-filtering algorithm tailored for that particular application, where a good compromise concerning computational complexity, training period and performance in terms of bit error rate can be reached.



(a) Accumulated bit error



(b) Square error

Figure 11.17 Experiments with DFE radial basis function algorithm, noise level -25dBs.

11.7 REFERENCES

1. V. J. Mathews, “Adaptive polynomial filters,” *IEEE Signal Processing Magazine*, vol. 8, pp. 10-26, Nov. 1991.
2. V. J. Mathews and G. L. Sicuranza, *Polynomial Signal Processing*, John Wiley & Sons, New York, NY, 2000.
3. J. Lee and V. J. Mathews, “A fast recursive least squares adaptive second-order Volterra filter and its performance analysis,” *IEEE Trans. on Signal Processing*, vol. 41, pp. 1087-1101, March 1993.
4. M. A. Syed and V. J. Mathews, “Lattice algorithms for recursive least squares adaptive second-order Volterra filtering,” *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, pp. 202-214, March 1994.
5. M. A. Syed and V. J. Mathews, “QR-decomposition based algorithms for adaptive Volterra filtering,” *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, pp. 372-382, June 1993.
6. V. J. Mathews, “Adaptive Volterra filters using orthogonal structures,” *IEEE Signal Processing Letters*, vol. 3, pp. 307-309, Dec. 1996.
7. J. Lee and V. J. Mathews, “A stability result for RLS adaptive bilinear filters,” *IEEE Signal Processing Letters*, vol. 1, pp. 191-193, Dec. 1994.
8. D. Williamson, R. A. Kennedy, and G. Pulford, “ Block decision feedback equalization,” *IEEE Trans. on Communications*, vol. 40, pp. 255-264, Feb. 1992.
9. B. Mulgrew, “Applying radial basis functions,” *IEEE Signal Processing Magazine*, vol. 13, pp. 50-65, March 1996.
10. F.-C. Zheng, S. McLaughlin, and B. Mulgrew, “Blind equalization of nonminimum phase channels: High order cummulant based algorithm,” *IEEE Trans. on Signal Processing*, vol. 41, pp. 681-691, Feb. 1993.
11. S. Chen, B. Mulgrew, and S. McLaughlin, “Adaptive Bayesian equalizer with decision feedback,” *IEEE Trans. on Signal Processing*, vol. 41, pp. 2918-2926, Sep. 1993.
12. S. Chen, S. McLaughlin, B. Mulgrew, and P. M. Grant, “Adaptive Bayesian decision feedback equalizer for dispersive mobile radio channels,” *IEEE Trans. on Communications*, vol. 43, pp. 1937-1946, May 1995.
13. S. Chen, B. Mulgrew, and P. M. Grant, “A clustering technique for digital communication channel equalization using radial basis function networks,” *IEEE Trans. on Neural Networks*, vol. 4, pp. 570-579, July 1993.
14. S. Chen and B. Mulgrew, “Reconstruction of binary signals using an adaptive-radial-basis function equaliser,” *Signal Processing*, vol. 22, pp. 77-93, Jan. 1991.
15. S. Chen and B. Mulgrew, “Overcoming co-channel interference using an adaptive radial basis function equaliser,” *Signal Processing*, vol. 28, pp. 91-107, July 1992.

16. S. Chen, S. McLaughlin, and B. Mulgrew, “Complex valued radial basis function network, Part I: Network architecture and learning algorithms,” *Signal Processing*, vol. 35, pp. 19-31, Jan. 1994.

17. S. Chen, S. McLaughlin, and B. Mulgrew, “Complex valued radial basis function network, Part II: Application to digital communications channel equalisation,” *Signal Processing*, vol. 36, pp. 175-188, March 1994.

18. I. Cha and S. Kassam, “Channel equalization using adaptive complex radial basis function networks,” *IEEE Trans. on Selected Areas in Communications*, vol. 13, pp. 122-131, Jan. 1995.

19. I. Cha and S. Kassam, “Interference cancellation using radial basis function networks,” *Signal Processing*, vol. 47, pp. 247-268, Dec. 1995.

20. D. Gonzaga, M. L. R. de Campos, and S. L. Netto, “Composite squared-error algorithm for training feedforward neural networks,” *Proc. of the 1998 IEEE Digital Filtering and Signal Processing Conference*, Victoria, B.C., June 1998.

21. B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1985.

22. A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw Hill, New York, NY, 3rd edition, 1991.

23. S. U. Qureshi, “Adaptive equalization,” *Proceedings of the IEEE*, vol-73, pp. 1349-1387, Sept. 1985.

24. B. Widrow and E. Walach, *Adaptive Inverse Control*, Prentice Hall, Englewood Cliffs, NJ, 1996.

25. K. K. Johnson and I. W. Sandberg, “Notes on the stability of bilinear filters,” *IEEE Trans. on Signal Processing*, vol. 46, pp. 2056-2058, July 1998.

26. F.-L. Luo and R. Unbehauen, *Applied Neural Networks for Signal Processing*, Cambridge University Press, Cambridge, U.K, 1996.

27. S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1999.

28. L.-X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1994.

29. C. Nikias and A. P. Petropulu, *Higher-order Spectra Analysis: A Nonlinear Signal Processing Framework*, Prentice Hall, Englewood Cliffs, NJ, 1993.

11.8 PROBLEMS

1. Perform the equalization of a nonlinear channel described by the following relation

$$r(k) = 0.9x(k) + 0.1x^2(k) - 0.3x^3(k) + n(k)$$

   using a known training signal that consists of a binary (-1,1) random signal. An additional Gaussian white noise with variance 10^{-2} is present at the channel output.
   Apply the LMS and RLS Volterra series algorithms.

2. Repeat problem 1 using the adaptive bilinear structure.

3. Repeat problem 1 using the multilayer perceptron algorithm.

4. Repeat problem 1 using the adaptive radial basis function structure.

5. Utilize a DFE equalizer to problem 1, also using the LMS and RLS Volterra series algorithms, and comment on the results.

6. Compare the performances of Volterra LMS and RLS algorithms in the identification of the following system.

$$\begin{aligned} d(k) &= -0.76x(k) - 1.0x(k-1) + 1.0x(k-2) + 0.5x^2(k) \\ &\quad +2.0x(k)x(k-2) - 1.6x^2(k-1) + 1.2x^2(k-2) \\ &\quad +0.8x(k-1)x(k-2) + n(k) \end{aligned}$$

   The input signal is a uniformly distributed white noise with variance $\sigma^2_{n_x} = 0.1$, filtered by all-pole filter given by

$$H(z) = \frac{z}{z - 0.95}$$

   An additional Gaussian white noise with variance 10^{-2} is present at unknown system output.

7. Identify an unknown system with the following model

$$d(k) = -0.6d(k-1) + x(k) + 0.01x(k)d(k-1) + 0.02x(k-1)d(k-1) + n(k)$$

   using the bilinear algorithm. The additional noise is Gaussian white noise with variance $\sigma_n^2 = -20$dB. Use Gaussian white noise with unit variance as input.

8. Repeat problem 7 using the multilayer perceptron algorithm.

9. Identify a system with the following nonlinear input to output relation

$$\begin{aligned} d(k) &= -0.08x(k) - 0.15x(k-1) + 0.14x(k-2) + 0.055x^2(k) \\ &\quad +0.30x(k)x(k-2) - 0.16x^2(k-1) + 0.14x^2(k-2) + n(k) \end{aligned}$$

   The input signal is Gaussian white noise with variance $\sigma_x^2 = 0.7$, and the measurement noise is also Gaussian white noise independent of the input signal with variance $\sigma_n^2 = 0.01$.
   Apply the radial basis function algorithm.

10. Repeat problem 9 using the multilayer perceptron algorithm.