4-22-1992

NONLINEAR ADAPTIVE SIGNAL PROCESSING

S.-W. Deng
Purdue University School of Electrical Engineering

O.K. Ersoy
Purdue University School of Electrical Engineering

NONLINEAR ADAPTIVE SIGNAL PROCESSING

S.-W. Deng and O.K. Ersoy

School of Electrical Engineering

Purdue University

West Lafayette, Indiana 47907-1285

TABLE OF CONTENTS

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

Deng, Shi-Wee. Ph.D., Purdue University, May 1992. Nonlinear Adaptive Signal Processing. Major Professor: Okan K. Ersoy.

Nonlinear techniques for signal processing and recognition have the promise of achieving systems which are superior to linear systems in a number of ways such as better performance in terms of accuracy, fault-tolerance, resolution, highly parallel architectures and closer similarity to biological intelligent systems. The nonlinear techniques proposed are in the form of multistage neural networks in which each stage can be a particular neural network and all the stages operate in parallel. The specific approach focused upon is the parallel, self-organizing, hierarchical neural networks (PSHNN's). A new type of PSHNN is discussed such that the outputs are allowed to be continuous-valued. The performance of the resulting networks is tested in problems of prediction of speech and of chaotic time-series. Three types of networks in which the stages are learned by the delta rule, sequential least-squares, and the backpropagation (BP) algorithm, respectively, are described. In all cases studied, the new networks achieve better performance than linear prediction. This is shown both theoretically and experimentally. A revised BP algorithm is discussed for learning input nonlinearities. The advantage of the revised BP algorithm is that the PSHNN with revised BP stages can be extended to use the sequential least-squares (SLS) or the least mean absolute value rule (LMAV) in the last stage.

A forward-backward training algorithm for parallel, self-organizing hierarchical neural networks is described. Using linear algebra, it is shown that the

forward-backward training of an n-stage PSHNN until convergence is equivalent to the pseudo-inverse solution for a single, total network designed in the least-squares sense with the total input vector consisting of the actual input vector and its additional nonlinear transformations. These results are also valid when a single long input vector is partitioned into smaller length vectors. A number of advantages achieved are small modules for easy and fast learning, parallel implementation of small modules during testing, faster convergence rate, better numerical error-reduction, and suitability for learning input nonlinear transformations by the backpropagation algorithm. Better performance in terms of deeper minimum of the error function and faster convergence rate is achieved when a single BP network is replaced by a PSHNN of equal complexity in which each stage is a BP network of smaller complexity than the single BP network.

CHAPTER 1
INTRODUCTION

1.1. Introduction

Linear signal processing is useful in many applications and relatively simple from conceptual and implementational view points, but there are still many applications in which nonlinear techniques of signal processing are effective. Nonlinear filters are very useful in modeling biological phenomena [KaPo85], myoelectrical signal processing [JaMF84], image processing and several other areas [AgEr91]. The method of adaptive polynomial filters which use Volterra series expansion was discussed by Mathews [Math91]. The Volterra filters with large enough order terms can approximate complex nonlinear systems; the disadvantage is large computational complexity and training time. Some neural networks can be characterized as nonlinear adaptive filters. Using neural networks, one can reduce the computational and the implementational complexity of adaptive polynomial filters. In this thesis, the spec'ific approach focused upon for the purpose is the parallel, self-organizing, hierarchical neural networks.

Parallel, self-organieing, hierarchical neural networks (PSHNN's) are multistage networks in which stages operate in parallel rather than in series during testing [ErHo90], [ErHoII]. The PSHNN is self-organizing in. the sense of number of stages. Each stage is a particular neural network referred to as the stage neural network (SNN). At the output of each SNN in previous PSHNN's, there is an error detection scheme which allows acceptance or rejection of input vectors. If an input vector is rejected, it goes through a nonlinear transformation before being inputted to the next stage. Only those input vectors which are rejected by present stage are fed into the next stage after nonlinear transformations. The PSHNN has many attractive properties. The experiments performed in comparison with backpropagation training indicated the superiority of the new architecture in the sense of classification accuracy, training time, parallelism and robustness [Hong90].

The PSHNN's as developed previously assumed quantieed or continuous-valued inputs and quantized, say, binary outputs. In this thesis, a new type of F'SHNN is discussed such that the outputs are allowed to be continuous-valued [ErDe911], [ErDe912]. In order to achieve this, all the input vectors are fed into all the stages after nonlinear transformations. The resulting networks are applied to the applications of predicting speech signals and simulating chaotic systems. The PSHNN's with continuous inputs and outputs are both theoretically and experimentally shown to make the square error sum (SES) smaller than that of linear filters [ErDe911], [ErDe912]. It is also shown that any input nonlinear transformation helps the system to achieve smaller SES than one-stage filters. During testing, the speed of processing with the PSHNN's are almost the same as with the one stage networks. In real applications, the square error sum we get by using the delta rule or backpropagation at each stage of the PSHNN is based on a suboptimal least-square solution. The suboptimal error reduction property is derived in Chapter 2. We find that the error reduction .property still holds when the delta rule is used [ErDe912].

Even though any kind of input nonlinearity guarantees better performance over a one-stage network, how to optimize the nonlinearities remain an open research issue. In this thesis, a revised backpropagation (RBP) network is proposed for learning input nonlinear transformations (NLT's) [ErDe912]. The RBP algorithm consists of two training steps, denoted as step I and step II, respectively. The RBP is the same as usual backpropagation [Rume88] during step I. During step II, we fix the weights between the input layer and the hidden layers, but retrain the weights between the last hidden and the output layers by the delta rule. There are several reasons why the RBP network may be preferable over the usual network with the BP algorithm. The first advantage is that the algorithm used during step II of RBP can be extended to satisfy other criteria such as the absolute error. The second reason is that the RBP algorithm allows faster learning. For this purpose the gain factor is chosen large for learning the input NLT during the first step, and the gain factor is reduced for fine training during the second step.

In adaptive signal processing, the sequential least-squares algorithm (SLS) allows each input sample to be used without the need for previous input samples [Grau84]. One advantage of the PSHNN with linear output nodes is that the SLS algorithm can be used [ErDe911]. This is generally not possible with other multistage neural networks. Sequential learning allows recursive updating of weight vectors in terms of the previous weight vectors, and the present input. For real-time signal processing, the SLS algorithm is essential. In Chapter 3, the PSHNN with the RBP stages and the SLS algorithm during step II is also discussed [DeEr922]. If a large block of N data points is being processed by the SLS or the least mean square (LMS) algorithm, we can choose the first K data points of the block ($K << N$) to learn the input NLT at each stage of the

PSHNN by the RBP. This technique can be repeated every N data points. In this way, short-time quasistationary signals like speech can be processed in real tirne.

In Chapter 2, we also discuss further error reduction in an n-stage network by circularly transmitting the remaining error through the stages a number of times until convergence [DeEr91]. Another important technique we propose in Chapter 4 is called the PSHNN with forward-backward training [DeEr921]. Asymptotic properties of the PSHNN with forward-backward training are discussed on a rigorous mathematical basis, in addition to providing additional experimental results. It is shown that the forward-backward training of an n-stage PSHNN until convergence is equivalent to the pseudo-inverse solution for a single, total network designed in the least-squares sense with the total input vector consisting of the actual input vector and its additional nonlinear transformations. These results are also valid when a single long input vector is partitioned into smaller vectors. The suboptimal asymptotic properties of the PSHNN's due to the use of the delta rule are also proved in Chapter 4.

Among deterministic optimization techniques, there is a method called the coordinate-descent algorithm [Luen84]. Given a pth order weight vector $W=(w_1 w_2 \cdots w_p)$, descent with respect to the coordinate w_i rneans that one minimizes the cost function $f(W)$ with respect to w_i, with other weight values fixed. Thus, changes in the single weight w_i are allowed in seeking a new and better weight vector W. The convergence rate of the coordinate-descent algorithm is usually slower than steepest descent. There is a similar phenomenon when the PSHNN with forward-backward training is comparecl to a one-stage total network. If we divide the linear input vectors of length p into p segments, then we can use a pstage PSHNN with forward-backward training (each stage with only one weight). The convergence rate of p-stage PSHNN with forward-backward training is usually slower than the one-stage network with p inputs. The PSHNN with forward-backward training can divide input vector into arbitrary segments with arbitrary length segments. For example, in function-link net,-works with higher order terms, the input vector gets very long [Pao89]. Using the PSHNN, we divide the input vector into a number of segments. Then, we observe in many cases that the PSHNN with forward-backward training converges faster than the function-link networks without partitioning. Beside faster convergence rate, another advantage of the PSHNN's is that each stage is much easier to implement than the function-link networks without partitioning.

Other criteria like least mean absolute value (LMAV) is superior to mean square error (MSE) in some applications. The LMAV rule is robust to outliers in a data set [Bell87]. In Chapter 5, the algorithm used during step II of the RBP is extended to the incorporation of the LMAV rule [DeEr922]. We also illustrate another method which use the BP algorithm with forward-backward training to learn input NLT's of the PSHNN. In this case, the interconnection weights between the input and the hidden layers are allowed to change sweep by sweep. The error reduction property by forward-backward training stated in Chapter 4 is based on the fixed input NLT of each stage of the PSHNN in every sweep. The PSHNN with BP stages and forward-backward training has different input NLT at each stage and at every sweep. We show the reason why the error reduction property still holds for this method in Chapter 5. Using this technique of learning input NLT's, better performance in terms of deeper minimum of the error function and faster convergence rate is achieved when a single BP network is replaced by a PSHNN of equal complexity in which each stage is a BP network of smaller complexity than the single BP network.

1.2. Thesis Organization

This thesis consists of six chapters. Chapter 2 illustrates the background for the model of the PSHNN with continuous inputs and outputs. Error reduction property is discussed both with single and multivariate inputs and outputs. The suboptimal error reduction property due to the use of the delta rule in practise is proved. A revised BP algorithm is proposed for learning input NLT's. In Chapter 3, we focus on incorporation of sequential learning. The PSHNN with SLS algorithm during step II of the RBP is also discussed. We introduce an algorithm called the PSHNN with forward-backward training and prove the asymptotic properties, both with optimal and suboptimal least-squares, in Chapter 4. Chapter 5 illustrates other methods of learning input NLT's. The RBP with the LMAV rule and the PSHNN with BP stages and forward-backward training are discussed. Conclusions and further research issues are presented in Chapter 6.

CHAPTER 2

PARALLEL, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORKS WITH CONTINUOUS INPUTS AND OUTPUTS

2.1. Introduction

Parallel, self-organizing, hierarchical neural networks (PSHNN's) are multistage networks in which stages operate in parallel rather than in series during testing [ErHo90], [ErHoII]. The PSHNN's as developed previously assume quantized or continuous-valued inputs and quantized, say, binary outputs [ErDe911]. In this chapter, a new type of PSHNN is proposed such that the outputs are allowed to be continuous-valued. A revised backpropagation algorithm (RBP) is discussed for learning input nonlinear transformations (NLT's) [ErDe912]. In order to achieve this, all the input vectors are fed into all the stages after nonlinear transformations. The performance of the resulting network is studied in the application of predicting speech signal samples from past samples.

Given a linear discrete-time system, the object of linear prediction is to estimate the output sequence from a linear combination of tbe past input samples. There are several ways to compute LPC (linear predictive coding) coefficients. One way is to solve the autocorrelation equations to find the LPC coefficients [Pars86]. Another way is by using the linear delta rule learning algorithm in a one-stage network [Rume88].

The PSHNN is both theoretically and experimentally shown to make the mean square error (MSE) smaller than with linear prediction. It is also shown that any input nonlinear transformation helps the system to achieve smaller mean square error than the MSE with linear prediction. By implementing the PSHNN stages in parallel, the speed of processing with several stages is almost the same as with one stage.

The chapter consists of 7 sections. In Sec. 2.2, the system model with a univariate output signal is discussed. The error reduction properties of the system are proved in Sec. 2.3. The results are generalized to a multivariate output signal in Sec. 2.4. The suboptimal error reduction property due to the use of the delta rule is derived in Sec. 2.5. The experimental results testing the model and the theory of the preceding sections with speech data are discussed in Sec. 2.6. So far the input nonlinear transformations are assumed to be known and constant. In Sec. 2.7, we describe how to learn the input NLT's by a revised backpropagation (RBP) network. Simulation results of learning input NLT's by the RBP are also given in this section.

2.2. System Model with Univariate Output Signal

The new PSHNN architecture proposed is shown in Fig. 2.1. In this section, we will assume a single output. SNN(i) represents the ith stage neural network which is trained by using the delta rule as discussed below. X(n) is the input vector sequence, and d(n) is the desired output sequence. X'(n), Y(n) and Z(n) are obtained by nonlinear transformations NLT1, NLT2 and NLT3 of X(n), respectively. NLT1, NLT2 and NLT3 are all different.

After SNN1 is trained with the delta rule, the error signal is

$$e_1(n)=d(n)-o_1(n).$$

We use $e_1(n)$ as the desired output of SNN2, and $Y(n)$ as the input signal to train SNN2 by the delta rule. The error signal for the second stage is

$$e_2(n)=e_1(n)-o_2(n).$$

After SNN2 is trained, we use $e_2(n)$ as the desired output of SNN3 to train SNN3 by using the delta rule. This process of adding stages is continued until the final error is negligible with white noise properties. Assuming three stages, the final output is

$$o_f(n)=o_1(n)+o_2(n)+o_3(n).$$

The delta rule is identically used in all the stages. For example, in the first stage, the sum of squared error minized by the delta rule is given by

$$E=\sum_{i=1}^{n}[d(i)-o_1(i)]^2, \tag{2.1}$$

where

$$o_1(i)=\sum_{j=1}^{p}a_j x(i-j), \tag{2.2}$$

$a_1, a_2 \cdots, a_p$ are the weights to be learned.

First, SNN1 generates the output $o_1(n)$ corresponding to the input vector $X(n)=[x(n-1), x(n-2), \ldots, x(n-p)]$. The value of a_i, $(i=1,\ldots,p)$ is modified at each iteration according to

$$\Delta_k a_i=\eta(d(k)-o_1(k))x(i), \tag{2.3}$$

where η is the gain factor of SNN(i).

The iterations are continued until $\Delta_k a_i$ becomes negligible. The procedure described above for the first stage **also** applies to the succeeding stages. The final error signal $e_f(n)$ is

$$e_f(n) \triangleq d(n) - o_f(n) \tag{2.4}$$

with $o_f(n) = o_1(n) + o_2(n) + o_3(n)$.

In Fig. 2.1, it is observed that

$$o_1(n) = d(n) - e_1(n)$$

$$o_2(n) = e_1(n) - e_2(n)$$

$$o_3(n) = e_2(n) - e_3(n)$$

$$\Rightarrow e_f(n) = e_3(n). \tag{2.5}$$

Let the error vectors for the first, second, and third stages be the following :

$$e_1 = (e_1(1), e_1(2), \ldots, e_1(n)),$$

$$e_2 = (e_2(1), e_2(2), \ldots, e_2(n)),$$

$$e_3 = (e_3(1), e_3(2), \ldots, e_3(n)).$$

We define

$$||e_f||^2 = ||e_3||^2 \triangleq \langle e_3 \cdot e_3 \rangle.$$

We prove $||e_1||^2 \geq ||e_2||^2 \geq ||e_3||^2$ in sections 2.3 and 2.4.

2.3. Error Reduction

In order to prove the properties of error reduction, we will first consider a two-stage PSHNN as shown in Fig. 2.1, and then generalize the properties to **n** stages. Assuming m training input vectors of length p and **NLT1** to be the identity operator $(X(n) = X'(n))$, we define

$$X = \begin{bmatrix} x_1^t \\ x_2^t \\ \cdot \\ \cdot \\ \cdot \\ x_m^t \end{bmatrix}$$

$$Y = \begin{bmatrix} y_1^t \\ y_2^t \\ \cdot \\ \cdot \\ \cdot \\ y_m^t \end{bmatrix}$$

$$D = \begin{bmatrix} d_1 & d_2 & \cdots & d_m \end{bmatrix}^t$$

$$W_1 = \begin{bmatrix} a_1 & a_2 & \cdots & a_p \end{bmatrix}^t$$

$$W_2 = \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix}^t.$$

X and Y are m **X** p matrices. Each row of X or Y represents input vector of SNNl or **SNN2,** respectively. D is the desired output vector of length m. W_1 and W_2 are vectors of length p. W_1 and W_2 are the weight vectors of SNNl and **SNN2,** respectively. The elements $a_1, a_2 \cdot \cdot \cdot, a_p$ in W_1 are **actually** the LPC coefficients. Usually m is greater than p. Using the delta rule to train W_1 and W_2 corresponds approximately to finding the **least-squares** solution to the equation

$$XW_1 = D. \tag{2.6}$$

The **least-squares** solution is [Erso88]

$$\overline{W}_1 = X^+D\ , \tag{2.7}$$

where X^+ is the pseudo-inverse of X.

The output of SNN1 is o_1, which can be expressed as

$$o_1 = X\overline{W}_1 = XX^+D\ . \tag{2.8}$$

The error vector of SNN1 is

$$e_1 = D-o_1 = (I-XX^+)D. \tag{2.9}$$

We define $A \triangleq XX^+$, which is positive semidefinite [DuHa73]. A is known as the projection operator.

The squared error $||e_1||^2$ is given by

$$||e_1||^2 = e_1^t e_1 = D^t(I-A)^t(I-A)D. \tag{2.10}$$

Since **(I-A)** is symmetric and idempotent [Stra86],

$$||e_1||^2 = D^t(I-A)D. \tag{2.11}$$

For SNN2, the input vector matrix is Y, and the desired output vector is e_1. A similar derivation yields

$$YW_2 = e_1,$$

$$\overline{W}_2 = Y^+e_1,$$

Y^+ is **pseudo-inverse** of matrix Y, and therefore

$$o_2 = YY^+e_1 = Be_1,$$

where we define $YY^+ \triangleq B$, which is also positive **semidefinite**. Then,

$$e_2 = e_1-o_2 = (I-B)e_1,$$

$$||e_2||^2 = e_1^t(I-B)e_1, \tag{2.12}$$

since **(I-B)** is also symmetric and idempotent.

Because **B** is positive semidefinite, we have

$$\begin{aligned} ||e_2||^2 &= e_1^t(I-B)e_1 \\ &\leq ||e_1||^2. \end{aligned} \tag{2.13}$$

This reasoning can be continued to any number of stages. For example, we let Z be the input vector matrix to stage 3, and define $C \triangleq ZZ^+$ which is symmetric, ideinpotent and positive semidefinite. We conclude that

$$\begin{aligned} ||e_3||^2 &= e_2^t(I-C)e_2 \\ &\leq ||e_2||^2. \end{aligned} \tag{2.14}$$

From Eqs. (2.5), (2.6), (2.13), and (2.14), it follows that

$$||e_f||^2 = ||e_3||^2 \leq ||e_2||^2 \leq ||e_1||^2.$$

Let us again consider the two-stage PSHNN. We can improve the results discussed above further by forward-backward training of stages. After we have trained W_1 and W_2, we use $D'=o_1+e_2$ as our new desired signal to train W_1 and W_2 once more. The new trained weights for SNNl become

$$W'_1 = X^+(o_1+e_2),$$

So, the new output of SNNl is

$$o'_1 = A(o_1+e_2) = o_1 + Ae_2,$$

since A is the projection operator, o_1 is already in the space spanned by A, and thereby $Ao_1=o_1$. The new error signal at the output of SNNl is

$$e'_1 = D'-o'_1 = (I-A)e_2. \tag{2.15}$$

Then, we get

$$||e'_1||^2 = e_2^T(I-A)e_2, \tag{2.16}$$

$$\Rightarrow \quad ||e'_1||^2 \le ||e_2||^2. \tag{2.17}$$

The new desired output for SNN2 is e'_1+o_2. Following the same procedure, the error vector for this stage is

$$e'_2 = (I-B)e'_1. \tag{2.18}$$

And also,

$$||e'_2||^2 = e'^T_1(I-B)e'_1, \tag{2.19}$$

$$\Rightarrow \quad ||e'_2||^2 \le ||e'_1||^2. \tag{2.20}$$

From Eqs. (2.17) and (2.20), we conclude that

$$||e'_2||^2 \le ||e_2||^2. \tag{2.21}$$

Eq. (2.21) shows that we can make further error reduction by forward-backward training in which the desired output of each stage is modified as the previous output plus the remaining error from the previously trained stage, and the training with the delta rule is repeated. It is straightforward to generalize the procedure above for any number of stages.

2.4. System Model with Multivariate Output Signal

If the output signal d_i is not a scalar but a $n \times 1$ vector denoted as $\underline{d}_i$, then the desired output D becomes

$$D = \left[\underline{d}_1 \ \underline{d}_2 \ \cdots \ \underline{d}_m\right]^t.$$

W_1 and W_2 of Section 2.3 become

$$W_1 = \left[\underline{a}_1 \ \underline{a}_2 \ \cdots \ \underline{a}_p\right]^t,$$

$$W_2 = \left[\underline{b}_1 \ \underline{b}_2 \ \cdots \ \underline{b}_p\right]^t,$$

where $\underline{a}_i$ and $\underline{b}_i$ are vectors of length n.

Now, D is an m X n matrix. W_1 and W_2 are p X n matrices. Based on the same derivation as in Section 2.3, the output of SNNl is an output matrix O_1 which is ideally

$$O_1 = X\overline{W}_1 = XX^+D. \tag{2.22}$$

The error of SNNl is

$$E_1 = D-O_1 = (I-XX^+)D. \tag{2.23}$$

E_1 is an $m \times n$ matrix, and can be expressed as

$$E_1 = \left[\underline{e}_1 \ \underline{e}_2 \ \cdots \ \underline{e}_m\right]^t.$$

We can define square error sum of stage 1 (ERRl) as

$$||\underline{e}_1||^2+||\underline{e}_2||^2+ \cdots +||\underline{e}_m||^2 .$$

Therefore,

$$ERR1 = tr(E_1E_1^T) = tr(E_1^TE_1). \tag{2.24}$$

Similar to Eq. (2.10), we get

$$ERR1 = tr(D^T(I-A)D). \tag{2.25}$$

Let ERR2 be the square error sum of SNN2. Repeating the same procedure, we get

$$ERR2 = tr(E_1^T(I-B)E_1). \tag{2.26}$$

Since B is positive semidefinite, we conclude that

$$ERR2 \leq ERR1. \tag{2.27}$$

The procedure discussed above can be easily extended to any number of stages.

2.5. Suboptimal Error Reduction Property

Assuming a two-stage network, the square error sum $||\mathbf{e}_2||^2$ in Eq. (2.12), is based on the optimal least-squares solution for the second stage. The least-squares error vector $\mathbf{e}_2$ is in the null space of YY^t. Defining $\xi_{ls}\overset{\Delta}{=}||\mathbf{e}_2||^2$, Eq. (2.12) can be written as

$$\xi_{ls}=||(I-YY^+)\mathbf{e}_1||^2=||P_{N(YY^t)}\mathbf{e}_1||^2, \tag{2.28}$$

where $P_{N(YY^t)}$ is the projection matrix to the null space of YY^t.

In reality, the square error sum we get by using the delta rule is based on a suboptimal least-squares solution. The suboptimal square error sum denoted as $\hat{\xi}_{ls}$ can be expressed as [Alex86], [Hayk91]

$$\hat{\xi}_{ls}=m(\xi_{min}+\xi_{exc}), \tag{2.29}$$

where m denotes the number of input vectors. ξ_{min} is the minimum mean square error (MSE) by solving the normal equation

$$E[Y_N(n)Y_N(n)^t]W_N=E[e_1(n)Y_N(n)], \tag{2.30}$$

where $Y_N(n)=[y(n),y(n-1),\cdots,y(n-N+1)]^t$, and N denotes the number of weights of SNN2 of Fig. 2.1; ξ_{exc} is due to the actual LMS weights jitter, and is sometimes referred to as the excess MSE. If we assume the sequence $y(n)$ is stationary and ergodic, then $m\xi_{min}$ in Eq. (2.29) will gradually approach the optimal square error sum ξ_{ls} as m grows. Thus, approximating $m\xi_{min}$ by ξ_{ls}, Eq. (2.29) can be written as

$$\hat{\xi}_{ls}=\xi_{ls}+m\xi_{exc} \tag{2.31}$$

ξ_{exc} is proportional to gain η used in training. Choosing smaller η aclhieves better suboptimal square error sum $\hat{\xi}_{ls}$, but then the learning rate is slower. So, there is a trade-off involved in choosing the value of η.

We show below that the error reduction properties in **Sec.** 2.3 still hold **with** the square error sum $\hat{\xi}_{ls}$ based on a suboptimal least-squares solution. Referring to Eq. **(2.12)**, we let $\mathrm{col}[YY^t]$ denote the column space of $[YY^t]$ and

$$P_{col[YY^t]}=YY^+, \qquad (2.32)$$

where $P_{col[YY^t]}$ is the projection matrix to the column space of $[YY^t]$. Then, the output vector of the second stage based on the optimal least square-solutions is [HoKu71], [RaMi71]

$$o_2=P_{col[YY^t]}e_1. \qquad (2.33)$$

The output vector $\hat{o}_2$ based on the suboptimal least-squares solution W'_2 is

$$\hat{o}_2=YW'_2. \qquad (2.34)$$

Eq. (2.34) shows that $\hat{o}_2$ is in the column space of $[YY^t]$, since it is generated by the data matrix Y. Consequently, $\hat{o}_2$ can be written as

$$\hat{o}_2=P_{col[YY^t]}e_1+b\ , \qquad (2.35)$$

where the vector b also belongs to the column space of $[YY^t]$. This is **graphically** shown in Fig. 2.2. The magnitude of b can be written as

$$||b||=c||P_{col[YY^t]}e_1||\ , \qquad (2.36)$$

where c satisfies $0<c<1$ in practise since the delta rule is a good approximation to the **least-squares** solution. Thus, the error vector of **SNN2** is

$$\hat{e}_2=e_1-\hat{o}_2$$

$$=[I-P_{col[YY^t]}]e_1-b$$

$$=P_{N[YY']}e_1-b. \tag{2.37}$$

Since $P_{N[YY']}e_1$ and b are orthogonal to each other, the magnitude of $\hat{e}_2$ satisfies

$$||e_2||^2 \leq ||\hat{e}_2||^2 = ||P_{N[YY']}e_1||^2 + ||b||^2, \tag{2.38}$$

$$||\hat{e}_2||^2 \leq ||P_{N[YY']}e_1||^2 + ||P_{col[YY']}e_1||^2 = ||e_1||^2. \tag{2.39}$$

Thus, $||\hat{e}_2||^2$ is less than $||e_1||^2$ as long as c is less than 1, which is definitely true in practise.

2.6. Experimental Results

The theoretical results discussed above were tested in the application of speech prediction. For this purpose, **100** speech samples at the sampling rate of **10 Khz** were used to train and to test the network. A sliding window of length between 4 and **10** data points were used to predict the next signal value following the window.

Properly choosing the value of the gain factor η in Eq. 2.3 is important. If we choose η too small, the convergence speed is too slow, but choosing too large η makes network oscillate. After trying different values of the gain factor, it was found that using a value between **0.001** to **0.1** was reasonable. In our experiments, we did not use momentum term.

We started with a two-stage PSHNN. The pointwise nonlinear transformations used in the experiments were the following:

(A) SIGMOID 1 (Sig. I)

$$Y(x) = \frac{1}{1+e^{-x}}$$

(B) SIGMOID 2 (Sig. II)

$Y(x) = 2 \times \text{sigmoid}(x) - 1$

(C) THRESHOLD 1 (Th. I)

$y = 1$ if $x \geq 0$

$y = 0$ if $x < 0$

(D) THRESHOLD 2 (Th.II)

$y = 1$ if $x \geq 0$

$y = -1$ if $x < 0$

In the experiments, we first normalized the data in the range [-1, 1]. In all experiments, NLT1 of the first stage is the identity operator, and 100 iterations of training were used for the first stage.

Table 2.1 shows the results, with 10 weight values as a function of the four types of nonlinearities. We used η=0.001 in the case of Th.I, Th.II and Sig.I, and η=0.1 in the case of Sig.II. The second stage converged after 300 iterations with Th.I and Th.II, and 100 iterations with Sig.I and Sig.II. It is observed in Table 2.1 that the two-stage PSHNN is always better in error performance than the one-stage network, the best result being the case of Sig.I nonlinearity. It is also observed that there is negligible error reduction in the case of Sig.II. This is because the input data was normalized in the range [-1,1], and this causes X and Y to be almost the same in this range.

The comparative performances of the one-stage and two-stage networks as a function of the length nc of the sliding window are shown in Table 2.2. The input nonlinearity used was Th.II. It is observed that both networks reach maximal performance at about nc equal to 10. Again, in all cases, the two-stage network has better error performance. In these experiments, the number of iterations in the two stages were 100 and 300, respectively.

The experiments discussed above were extended to three stages, with nc=5 for each stage. The results are shown in Table 2.3. It is observed that further reduction of error depends on the combination of nonlinearities used. An important research issue is how to optimize the nonlinearities. An effective approach is by using the revised backpropagation (RBP) network discussed in the next section.

2.7. Learning Input Nonlinear Transformation by Revised Backpropagation

In the proceeding sections, it became clear that how to choose the input nonlinearities for optimal performance is an important issue. In this section, a revised backpropagation (RBP) network is proposed for this purpose.

The RBP network consists of linear input and output units and nonlinear hidden units. One hidden layer is often sufficient. The hidden layers represent the nonlinear transformation of the input vector. The output of the jth unit of the kth layer is of the form

$$O_k(j)=f\Big(\sum_{i=1}^{N_{k-1}} W_k(j,i)O_{k-1}(i)\Big) ,$$

where N_{k-1} is the number of output nodes of the (k-1)th layer; O_{k-1} is the output vector of the (k-1)th layer; $W_k(.,.)$ are the weights connecting the (k-1)th and the kth layers, and $f(.)$ is the nonlinear activation function, assumed to be differentiable and usually chosen monotone nondecreasing.

Fig. 2.3 is a two-stage PSHNN with RBP Stages. The RBP algorithm consists of two training steps, denoted as step I and step II, respectively. During step I, the RBP is the same as the usual backpropagation (BP) algorithm

[Rume88]. During step II, we fix the weights between the input layer and the hidden layers, but retrain the weights between the last hidden and the output layers by the delta rule.

Each stage of the PSHNN now consists of a RBP network, except possibly the first stage which can be learned by the delta rule alone, with NLT1 equal to the identity operator. In this way, the first stage can be considered as the linear part of the system.

There are a number of reasons why the two-step training described above is preferable over the usual training with the BP algorithm. The first reason is that it is possible to use the PSHNN with RBP stages together with the SLS algorithm or the delta rule. For this purpose, we assume that the signal is reasonably stationary for short time duration. Thus, the weights between the input and the hidden layers of the RBP stages can be kept constant during such a time window. Only the last stage of the RBP network is then made adaptive by the SLS algorithm or the delta rule, which is much faster than the BP algorithm requiring many sweeps over a data block.

The second reason is that the two-step algorithm allows faster learning. During the first step, the gain factor is chosen rather large for fast learning. During the second step, the gain factor is reduced for fine training. The end result is considerably faster learning than with the regular BP algorithm. It can be argued that the final error vector may not be as optimal as the error vector with the regular BP algorithm. We believe that this is not a problem since successive RBP stages compensate for the error. As a matter of fact, considerably larger errors, for example, due to imperfect implementation of the interconnection weights and nonlinearities can be tolerated due to error compensation [ErHoII].

The results of the computer experiments carried out with the same speech data are shown in Table 2.4. **In** these experiments, the length of the input vector was five; the gain factor was **1.0** in step I and **0.01** in step **II;** tbe number of iterations was **1000** in step I and **100** in step **II.** It is observed in Table 2.5 that the best performance is obtained with four bidden units. It is also observed that the error performance is considerably better than the results in the previous tables with fixed **NLT's.**

Table 2.1. Performance of One-Stage and Two-Stage PSHNN as a Function of Input Nonlinearities (errl = $||e_1||^2$, err2 = $||e_2||^2$).

type of NLT	square error sum	
	errl	err2
Th.I	1.3785	1.3500
Th.II	1.3785	1.3360
Sig.I	1.3785	1.1221
Sig.II	1.3785	1.3766

Table 2.2. Performance of One-Stage and Two-Stage PSHNN's as a Function of the Length of the Weight Vector When the Input Nonlinearity is Th.II (err1 = $||e_1||^2$, err2 = $||e_2||^2$).

nc	square error sum	
	err1	err2
4	2.1707	2.1151
5	2.1359	2.0813
6	2.1317	2.0721
7	1.7876	1.7465
8	1.4419	1.4058
9	1.3789	1.3345
10	1.3780	1.3346

Table 2.3. Performance of One-Stage, Two-Stage and Three-Stage PSHNN's as a Function of Input Nonlinearities (err1= $||e_1||^2$, err2= $||e_2||^2$, err3= $||e_3||^2$).

Type of NLT		square error sum			Number of Iterations		
Stage II	Stage III	err1	err2	err3	Stage I	Stage II	Stage III
Th.II	Sig.II	2.1357	2.1346	2.0986	100	500	100
Sig.II	Th.II	2.1357	2.0943	2.0789	100	500	100
Sig.I	Sig.II	2.1357	2.1346	2.1345	100	500	2000
Sig.II	Th.I	2.1357	2.1346	2.100	100	500	100

Table 2.4. Performance when the Input NLT is Learned by RBP ($err1=||e_1||^2, err2=||e_2||^2, err3=||e_3||^2, err4=||e_4||^2$).

Number of Hidden Nodes	2		3		4		5	
square error sum	step I	step II	step I	step II	step I	step II	step I	step 11
err1		2.1352		2.1352		2.1352		2.1352
err2	2.1369	2.1347	1.4857	1.4625	1.2191	1.1917	1.8991	1.8333
err3	2.1047	2.0974	1.1818	1.1357	1.1675	1.1697	1.6982	1.5758
err4	1.6779	1.6646	1.0795	1.0731	1.0164	0.9790	1.3681	1.3527

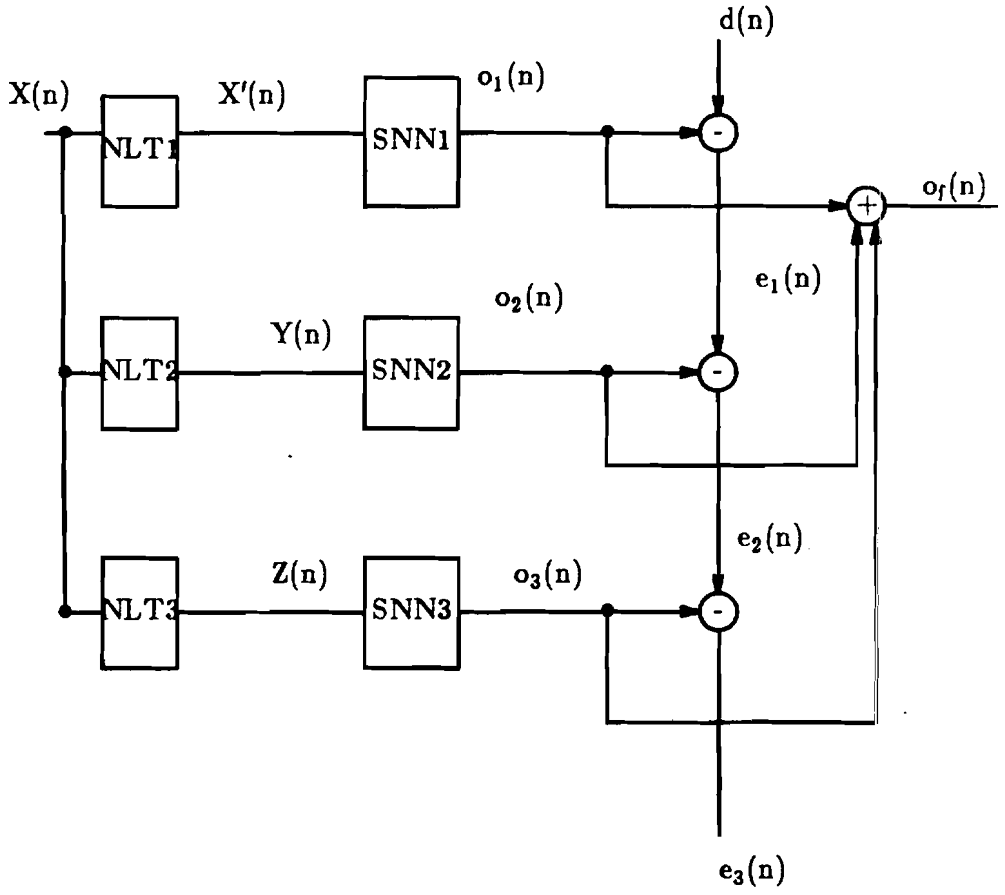

Figure 2.1. Block Diagram for a Three-Stage PSHNN.

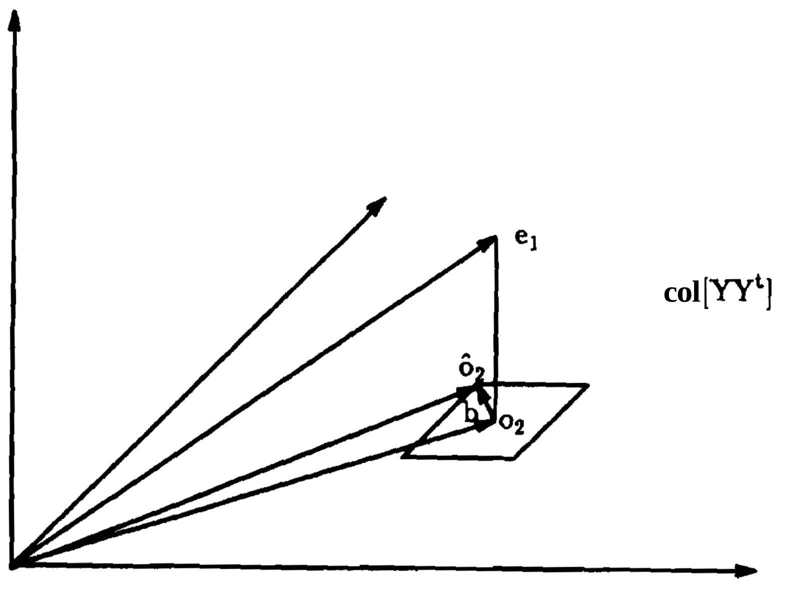

Figure 2.2. Representation of Suboptimal Solution.

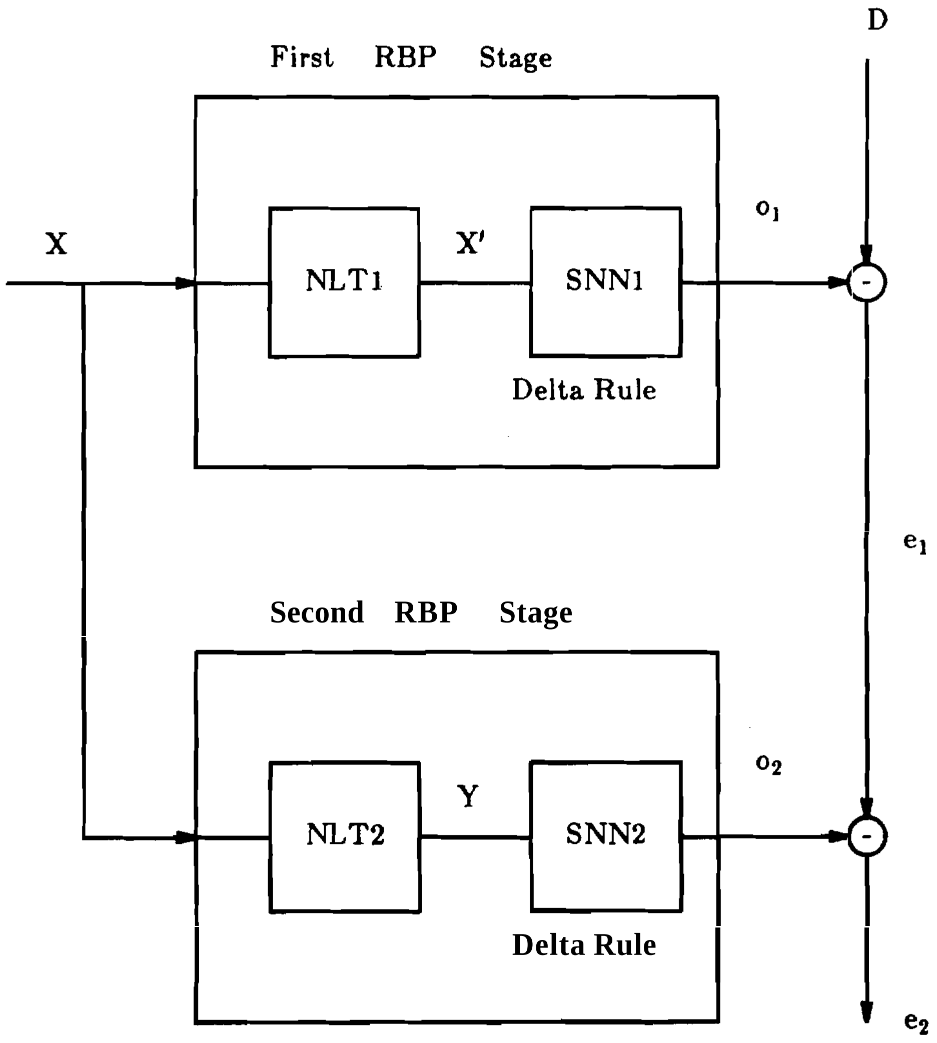

Figure 2.3. Two-Stage PSHNN with RBP Stages.

CHAPTER 3
INCORPORATION OF SEQUENTIAL LEAST-SQUARES

3.1. Introduction

One advantage of PSHNN is that the sequential **least-squares** (SLS) algorithm can be used for learning. This does not seem possible with other multistage neural networks.

The **least-squares** solution discussed in Chapter 2 is commonly referred to as batch processing **least-squares** because the data $\mathbf{D=(d_1 d_2 \ldots d_m)}$ are processed simultaneously [Sore85]. If new data $\mathbf{d_{m+1}}$ are to be processed after having determined an estimate based on the data D, it is necessary to completely reprocess the old data with previous neural networks. To avoid this inefficient procedure, we need to consider the determination of the **least-squares** estimate from an estimate based on D and the new data $\mathbf{d_{m+1}}$ without explicitly using D in PSHNN.

In adaptive signal processing, the SLS algorithm allows each input samples to be used without the need for previous input samples. In real-time adaptive signal processing, it is not possible to use a batch method with long training time, and the SLS algorithm is essential. In this chapter, the algorithm used during step II of the RBP is extended with the incorporation of the SLS. In this way, the **RBP** networks with the SLS can be used to process **short-time** stationary signals in real time.

The chapter consists of 4 sections. In Sec. 3.2, the PSHNN with the SLS algorithm is discussed. The RBP network with the SLS is proposed in Sec. 3.3. Experimental results are provided in Sec. 3.4.

3.2. Incorporation of Sequential Learning

In Chapter 2, we found optimal solutions for the weight vectors in terms of the generalized inverse of the input data matrix X. Sequential learning allows recursive updating of weight vectors in terms of the previous weight vectors, and the present input. In this way, it is not necessary to store past data vectors in memory.

It can be shown that the SLS algorithm reduces to the following set of two recursive equations [Kell90] [Grau84].

$$W_1(r) = W_1(r-1) + P_r X_r (x_r - X_r^T W_1(r-1)) , \tag{3.1}$$

$$P_r = P_{r-1} - \frac{P_{r-1} X_r X_r^T P_{r-1}}{1 + X_r^T P_{r-1} X_r} . \tag{3.2}$$

Here X_r is the column vector containing the input signals x_{r-1} to x_{r-p}, r is an index representing the current input signal, and p is the number of LPC coefficients. $W_1(r)$ is the present estimate of LPC coefficients expressed as a column vector, and $W_1(r-1)$ is the previous estimate of this vector at time r-1. P_r is a p$\times$p matrix which corresponds to the rth iteration. The value of P_r can be calculated recursively by Eq.(3.2). Initially, $W_1(0)$, which is a column vector, is zeroed, and the matrix P_0 is set equal to some constant product of the p by p identity matrix [Mend73].

For SNN2, we replace X_r by Y_r, and the recursive SLS equations are

$$W_2(r) = W_2(r-1) + P_r Y_r (e_1(r) - Y_r^T W_2(r-1)) \,, \tag{3.3}$$

$$P_r = P_{r-1} - \frac{P_{r-1} Y_r Y_r^T P_{r-1}}{1 + Y_r^T P_{r-1} Y_r} \,. \tag{3.4}$$

Here $e_1(r)$ is the error signal for the SNN1 at the present time, given by

$$e_1(r) = x_r - o_1(r) \,.$$

For SNN3, we replace X_r by Z_r, and get

$$W_3(r) = W_3(r-1) + P_r Z_r (e_2(r) - Z_r^T W_3(r-1)) \,, \tag{3.5}$$

$$P_r = P_{r-1} - \frac{P_{r-1} Z_r Z_r^T P_{r-1}}{1 + Z_r^T P_{r-1} Z_r} \,. \tag{3.6}$$

Where $e_2(r) = e_1(r) - o_2(r)$.

The final output is

$$o_f = o_1 + o_2 + o_3 \,.$$

3.3. The RBP Networks with the SLS Algorithm

We have discussed the revised backpropagation (RBP) algorithm in Chapter 2. Referring to Fig. 3.1, the RBP network with the SLS uses the sequential least-squares during step II of the RBP algorithm. Thus, the weights between the input and the hidden layers of the RBP stages can be kept constant during such a time window. Only the last stage of the RBP network is made adaptive by the SLS algorithm, which is much faster than the BP algorithm requiring many sweeps over a data block. For this purpose, we assume that the signal is reasonably stationary for N data points. While the block of N data points is being processed with the SLS algorithm, the first M << N data points of the block can be used to train the stages of the PSHNN by the BP algorithm. At the start of the next time window of N data points, the RBP stages are renewed with the new weights between the input and the hidden layers of the RBP stages. This process is repeated periodically every N data points. In this way, nonstationary signals which can be assumed to be stationary over short time intervals can be effectively processed.

3.4. Experimental Results

We experimented with two-stage PSHNN's using the SLS learning algorithm. The nonlinear transformations used in the experiments are the same as in Chapter 2. The error performance results are shown in Tables 3.1 and 3.2. Previous conclusions are again valid in this case. Another observation is that it is necessary to optimize the networks both in terms of the length of the weight vectors and the number of stages.

Fig.3.2 through Fig.3.4 show the prediction results with sequential learning. The prediction was started after 7 initial speech samples. Nonlinearity of Th.II was used and the length of the weight vector was 7. Figs. 3.2 and 3.3 show the original speech signal versus the predicted speech signal with one-stage and two-stage networks, respectively. Fig.3.4 shows the prediction error with the same networks. These results show that the two-stage network with SLS learning has better prediction performance than the traditional one-stage network with SLS learning. Since the two stages are implemented in parallel, the gains are achieved with almost the same processing time as the one-stage network.

The simulations in Table 3.3 and Table 3.4 used a RBP stage with the SLS rule in place of the second stage of the PSHNN of the previous experiments. In these two simulations, the RBP networks had 5 input units, and 1 output unit; five hidden nodes were used in Table 3.3 and four hidden nodes in Table 3.4. The gain factors used during step I were 0.5 in Table 3.3 and 1.0 in Table 3.4. Tables 3.3 and 3.4 show that the performance of learning input NLT2 by the RBP stage is better than any pointwise NLT2.

Figs. 3.5 thru 3.7 show the prediction results with sequential learning. The prediction was started after 5 initial speech samples. Th.II was used as the nonlinearity and the length of the sliding window was 5. Figs. 3.5 and 3.6 show the original speech signal versus the predicted speech signal with the one-stage and the two-stage networks, respectively. Fig. 3.7 shows the prediction error with both networks. These results also show that the two-stage network with SLS learning has better prediction performance than the traditional one-stage network with SLS learning. Fig. 3.8 shows the original versus the predicted signals of the two-stage PSHNN with the RBP and the SLS rule in the second stage and 1000 iterations used during step I of RBP. Fig. 3.9 shows the predicted error of the two-stage network with Th. II pointwise NLT2 versus the predicted error of the two-stage network with the RBP and the SLS rule in the second stage.

Table 3.1. **Nonlinear Speech Prediction Performance of One-Stage and Two-Stage PSHNN's Trained with SLS Learning** (nc=7, errl= $)|e_1|)$*, err2= $||e_2||^2$).

type of NLT	square error sum	
	errl	err2
Th.I	**1.3255**	**1.2960**
Th.II	**1.3255**	**1.2856**
Sig.I	**1.3255**	**1.3221**
Sig.II	**1.3255**	**1.3252**

Table 3.2. Nonlinear Speech Prediction Performance of One-Stage and Two-Stage PSHNN's Trained with SLS Learning (nc=5, err1= $||e_1||^2$, err2= $||e_2||^2$).

type of NLT	square error sum	
	err1	err2
Th.I	1.7799	1.7332
Th.II	1.7799	1.7197
Sig.I	1.7799	1.7783
Sig.II	1.7799	1.7797

Table 3.3. Performance of a 5 Hidden Unit Two-Stage PSHNN with the RBP and the SLS Rule in the Second Stage.

# of training	square error sum	
	step I	step II
500	1.4783	1.4711
600	1.4042	1.4002
700	1.3387	1.3360
800	1.2748	1.2718
900	1.1935	1.1903
1000	1.1189	1.1178

Table 3.4. Performance of a 4 Hidden Unit Two-Stage PSHNN with the RBP and the SLS Rule in the Second Stage.

# of iterations	square error sum	
	step I	step II
500	1.4314	1.4099
600	1.3272	1.2180
700	1.1170	1.0130
800	0.9646	0.9221
900	0.9148	0.8842
1000	0.8850	0.8568

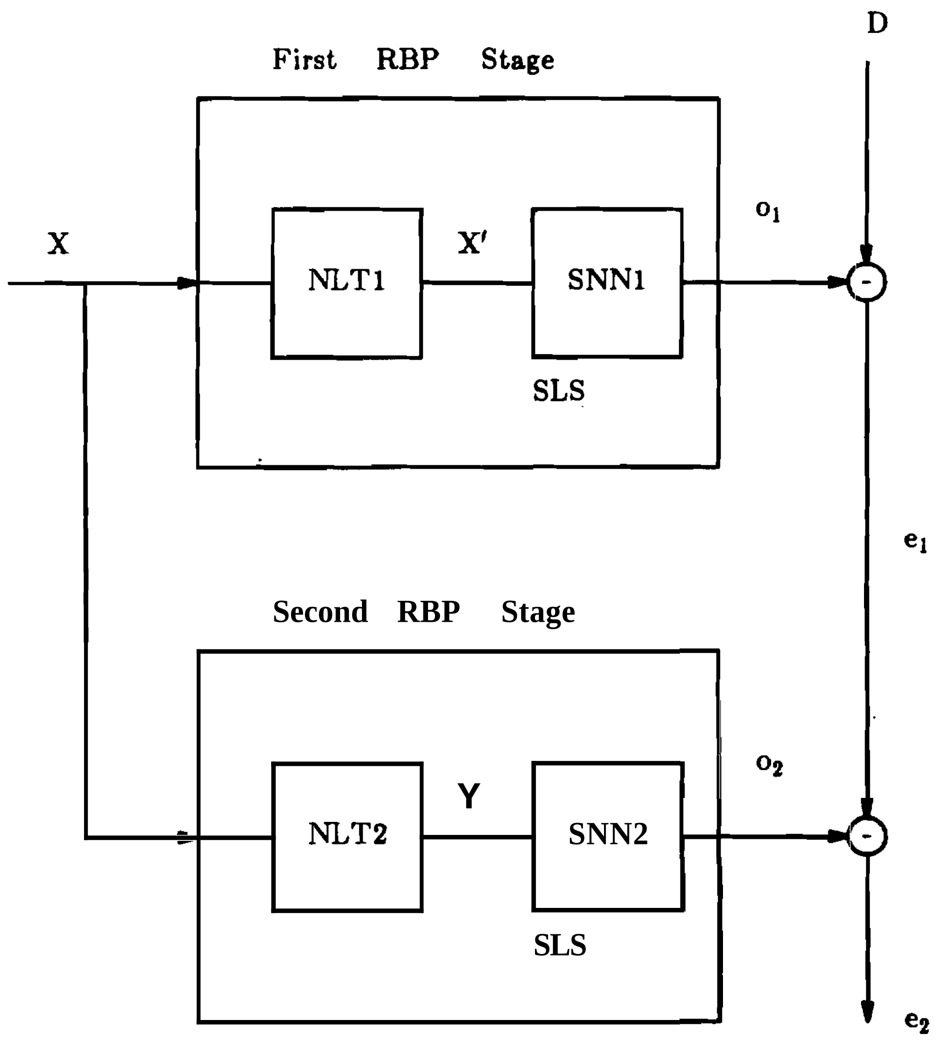

Figure 3.1. Two-Stage PSHNN with RBP Stages and the SLS Algorithm.

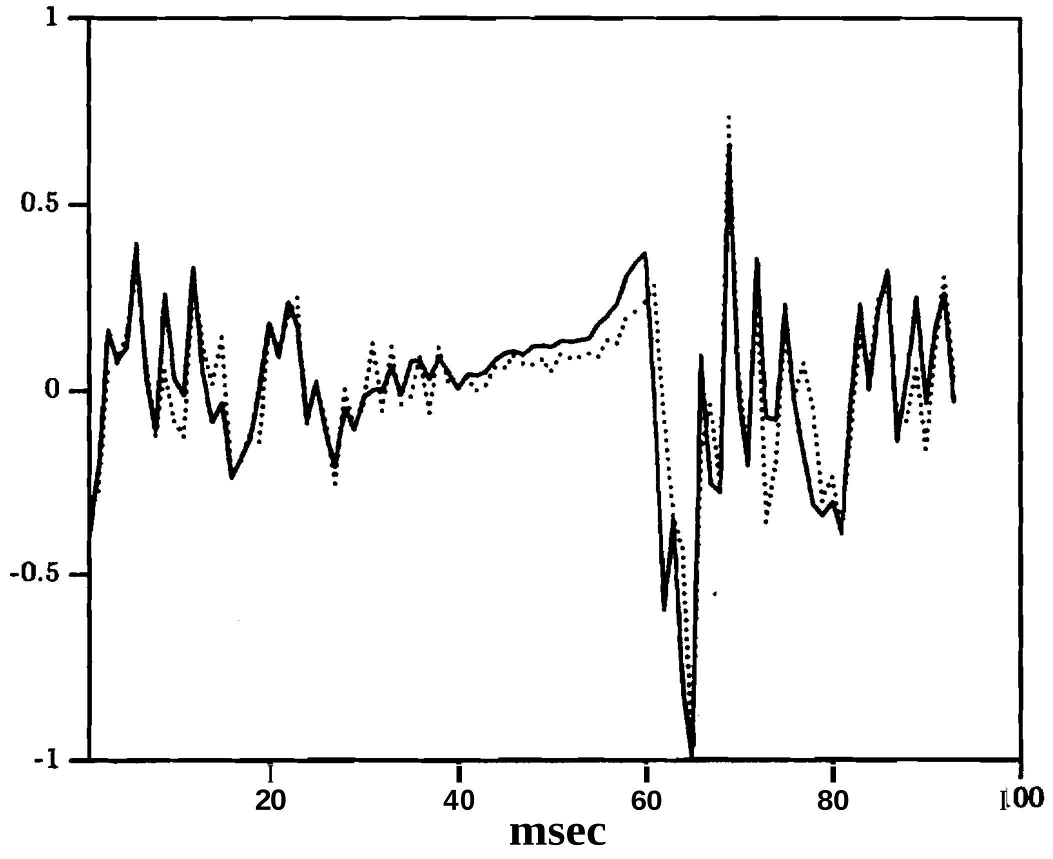

Figure 3.2. Original Speech Signal (solid line) and the Predicted Speech Signal (dotted line) with One-Stage HNN Trained with the SLS Algorithm.

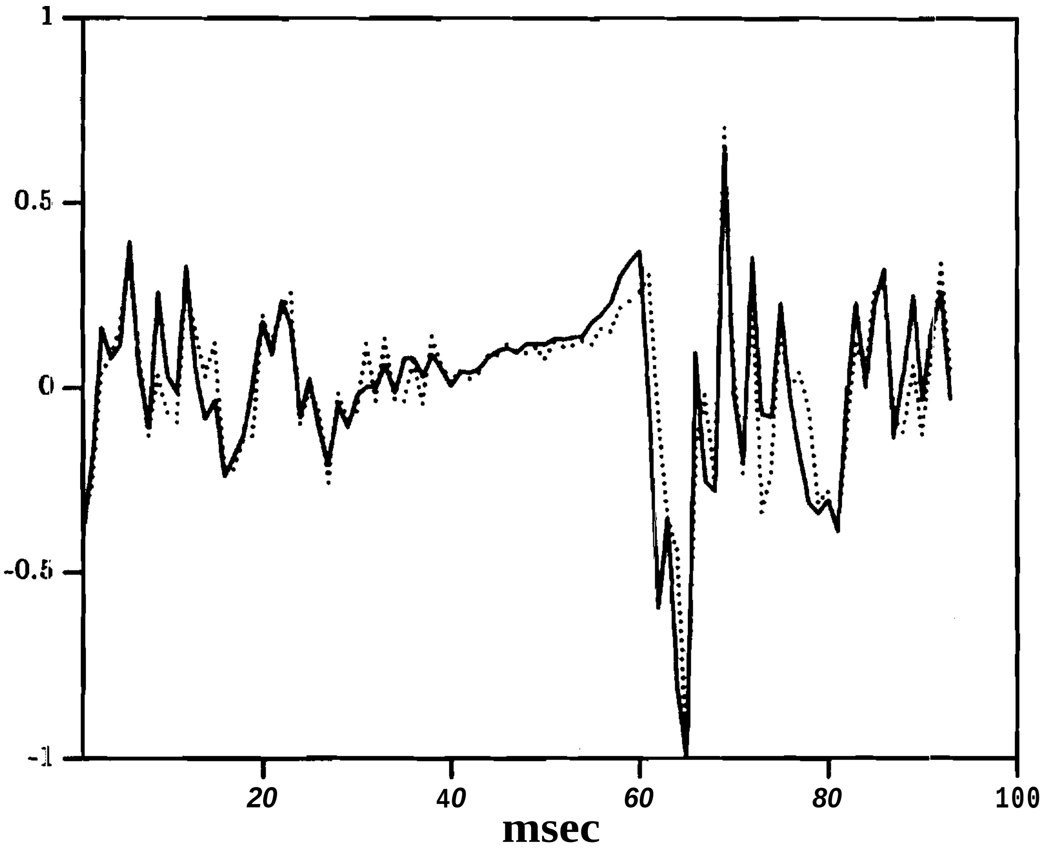

Figure 3.3. Original Speech Signal (solid line) and the Predicted Speech Signal (dotted line) with Two-Stage HNN Trained with the SLS Algorithm.

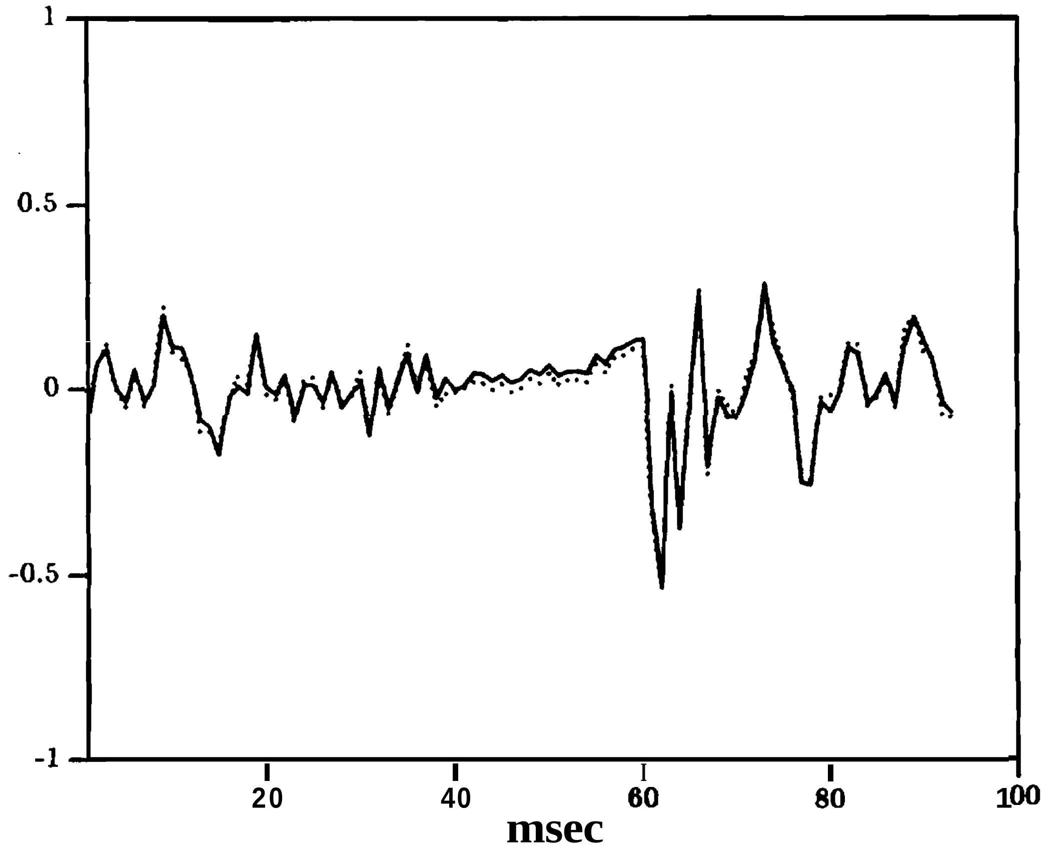

Figure 3.4. The Error Signals with One-Stage HNN (solid line) and Two-Stage HNN (dotted line) Trained with the SLS Algorithm.

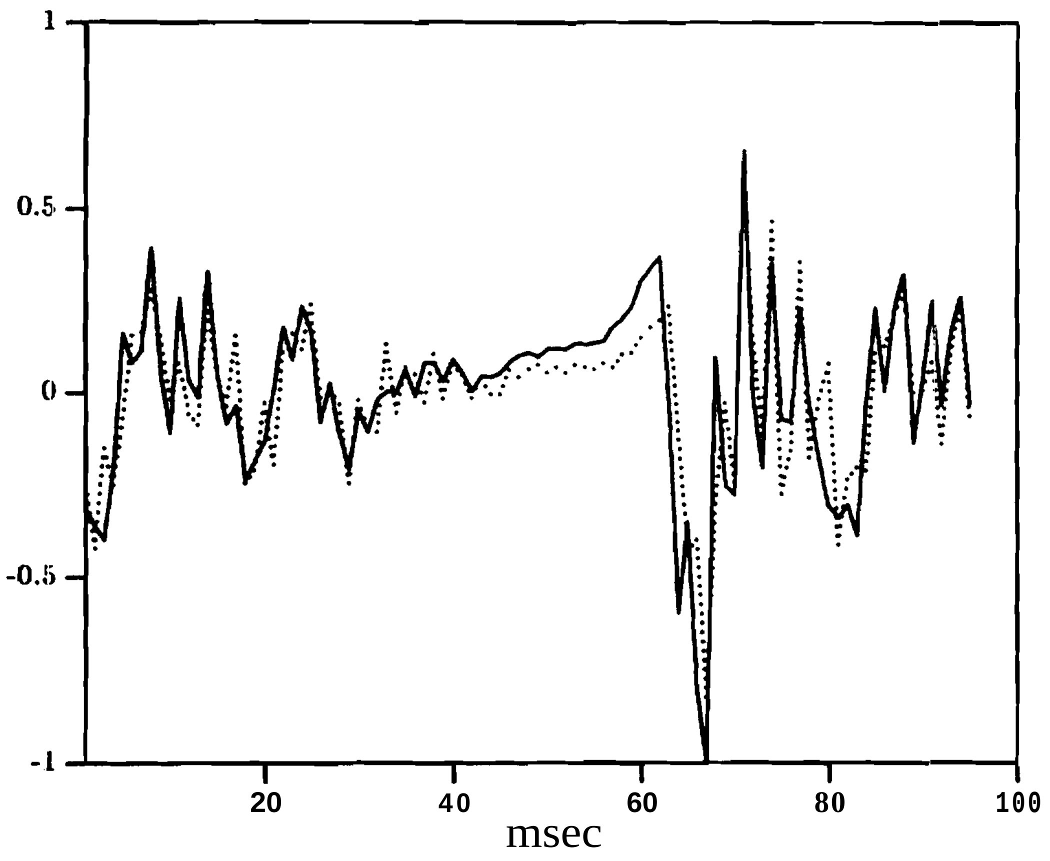

Figure 3.5. Original Speech Signal (Solid Line) and the Predicted Speech Signal (Dotted Line) with One-Stage PSHNN Trained with the SLS Algorithm (nc=5).

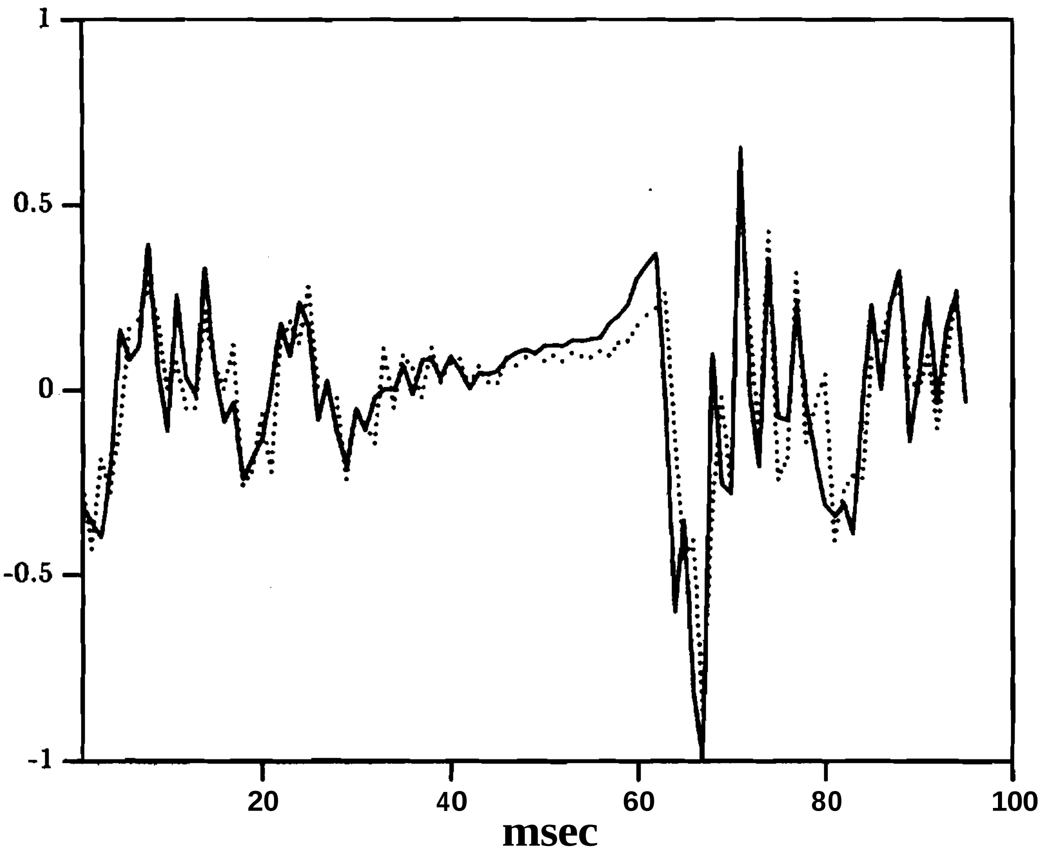

Figure 3.6. Original Speech Signal (Solid Line) and the Predicted Speech Signal (Dotted Line) with Two-Stage PSHNN Trained with the SLS Algorithm (nc=5).

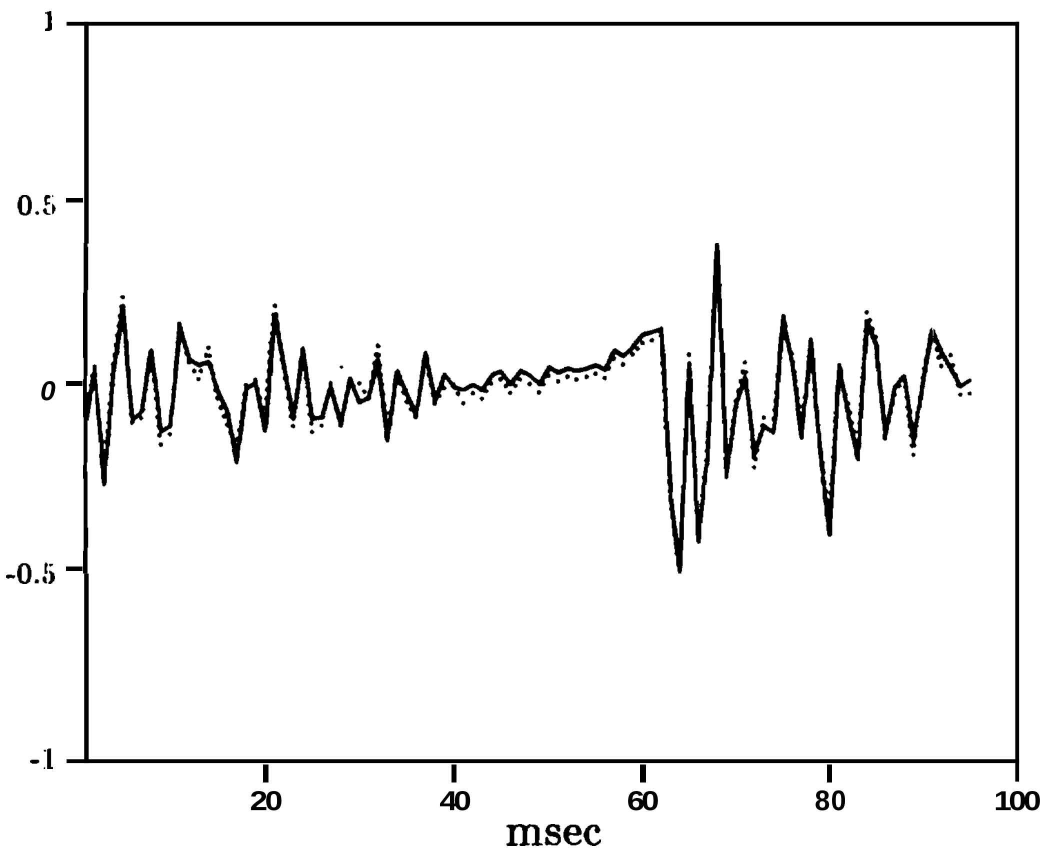

Figure 3.7. The Error Signals with One-Stage PSHNN (Solid Line) and Two-Stage PSHNN (Dotted Line) Trained with the SLS Algorithm (nc=5).

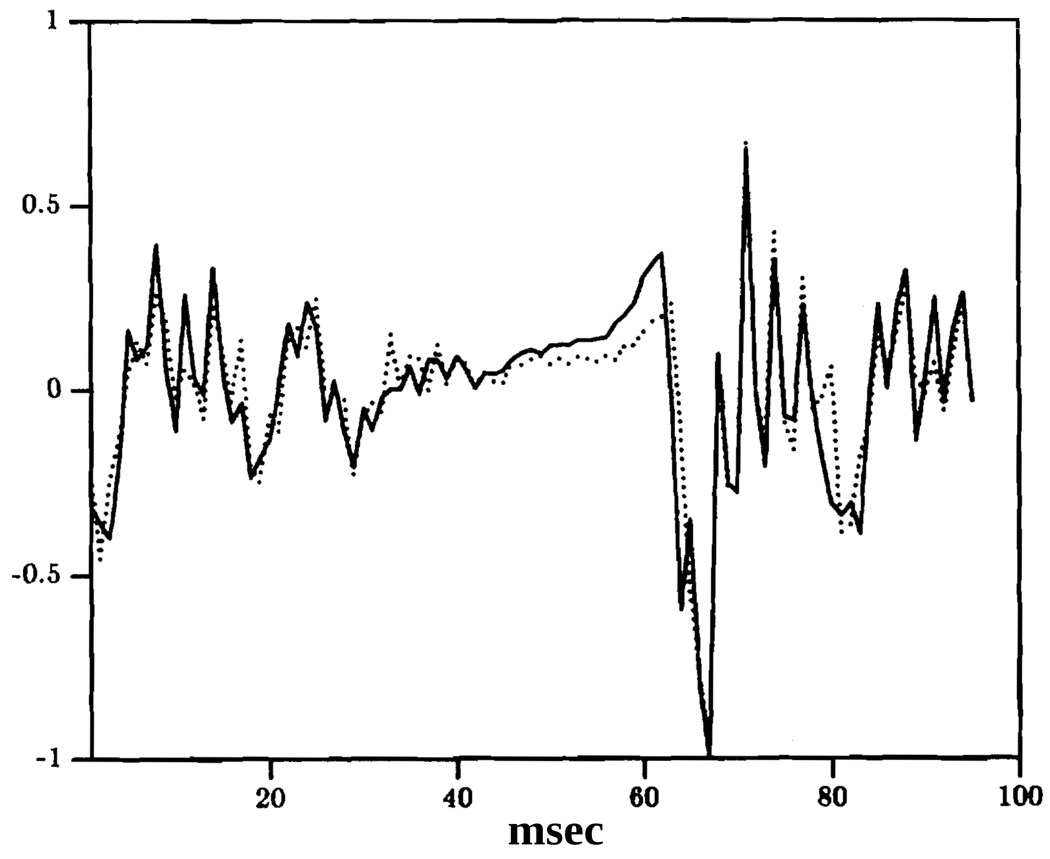

Figure 3.8. Original Speech Signal (Solid Line) and the Predicted Speech Signal (Dotted Line) with Two-Stage PSHNN with the RBP and SLS Rule on the Second Stage (nc=5).

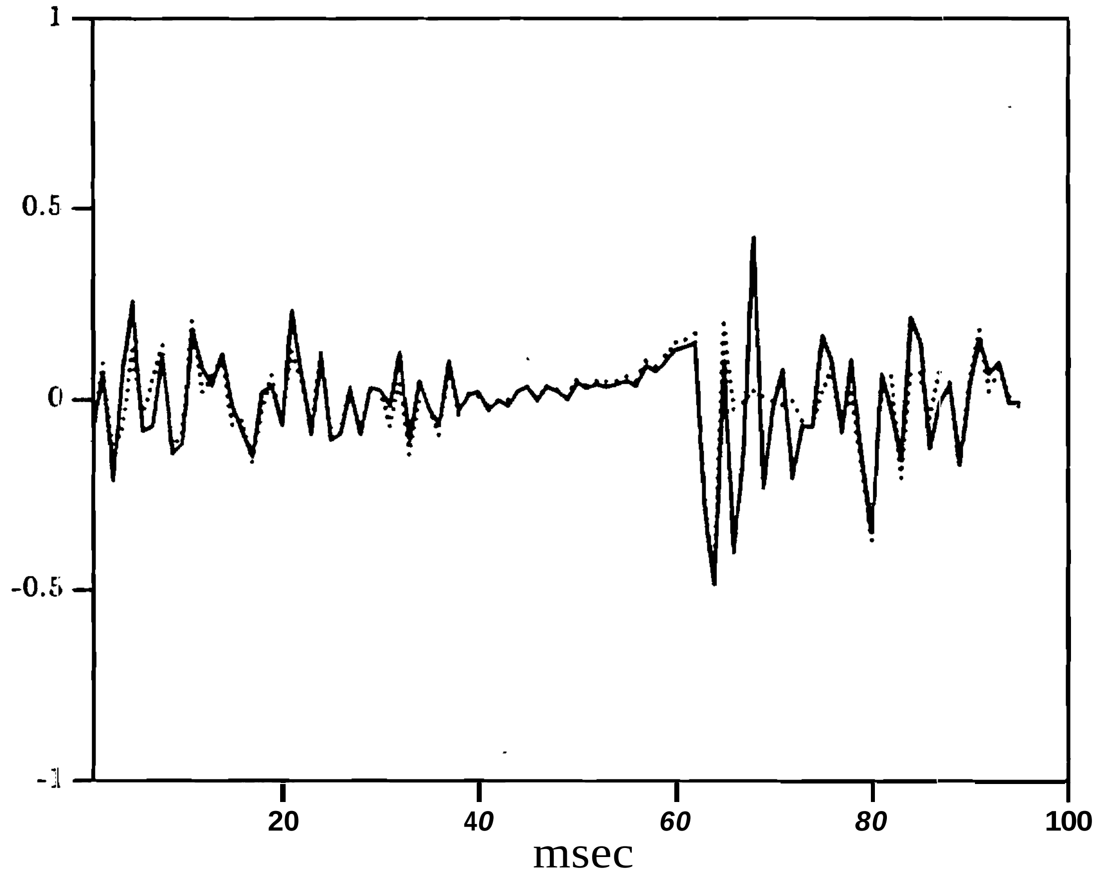

Figure 3.9. The Error Signals with Two-Stage PSHNN (Solid Line) with NLT2=Th.II and Two-Stage PSHNN (Dotted Line:) with the RBP and the SLS Rule on the Second Stage (nc=5).

CHAPTER 4

PARALLEL, SELF-ORGANIZING, HIERARCHICAL N E W NETWORKS WITH FORWARD-BACKWARD TRAINING

4.1. Introduction

In Chapter 2, we discussed the generalization of parallel, self-organizing, hierarchical neural networks (PSHNN's) to continuous inputs as well as continuous outputs [ErDe912]. The block diagram for such a 3-stage PSHNN is shown in Fig. 2.1. It was shown that the stages are generated by nonlinearly transforming input vectors, and each new stage attempts to correct the errors of the previous stage. It was also discussed that further error reduction in an n-stage network is possible by circularly transmitting the remaining error through the stages a number of times until convergence. Running through all the stages once can be called one sweep. At each successive sweep, the desired output of each stage is modified as the previous output of the stage plus the remaining error from the previous stage. The first stage receives the error from the last stage. Both in Ref. [ErDe912] and in this Chapter, the output nodes are assumed to be linear.

In this chapter, forward-backward training of n-stage PSHNN's are introduced and discussed on a rigorous mathematical basis, in addition to providing experimental results. The results are actually valid for all linear least-squares problems if we consider the input vector and the vectors generated from it by nonlinear transformations as the decomposition of a single, long vector. In this sense, the techniques discussed represent the decomposition of a large problem into smaller problems which are related through errors and forward-backward training [DeEr921]. Generation of additional nodes at the input is common to a number of techniques such as generalized discriminant functions [DuHa73], higher order networks [GiMa87], and function-link networks [Pao89]. After this is done, a single total network can be trained by the delta rule [WiHo60]. At convergence, the result is approximately the same as tbe pseudo-inverse solution, disregarding any possible numerical problems [ErDe912]. The PSHNN's are different because the single total network are replaced by a number of subnetworks.

The main result in this chapter is that forward-backward training of an n-stage network until convergence is equivalent to the pseudo-inverse solution for a single total network with the total number of input nodes if each stage is optimized in the sense of least-squares. There are a number of advantages in achieving the pseudo-inverse solution in this fashion. The most obvious advantage is that each stage is much easier to implement as a module to be trained than the whole network. In addition, all stages can be processed in parallel during testing. If the complexity of implementation without parallel stages is denoted by f(N) where N is the length of input vectors, the parallel complexity of the forward-backward training algorithm during testing is f(K) where K equals N/M with M equal to the number of stages.

The chapter consists of six sections. In Sec. 4.2, the forward-backward training algorithm is described in detail. In Sec. 4.3, the asymptotic properties with a two-stage network are discussed. These properties are extended to n-stage networks in Sec. 4.4. The suboptimal asymptotic properties due to the use of the delta rule during training are proved in Sec. 4.5. Experimental results are provided in Sec. 4.6.

4.2. PSHNN with Forward-Backward Training

The system model is shown in Fig. 2.1. In this section, a single output is assumed. In Fig. 2.1, SNN(i) represents the i-th stage neural network. In this chapter, the stage neural network is assumed to be trained by the delta rule [Rume88]. The output nodes are assumed to be linear. X(n) is the input vector sequence; d(n) is the desired output sequence; X'(n), Y(n) and Z(n) are obtained by different nonlinear transformations NLT1, NLT2 and NLT3.

We first consider a two-stage PSHNN, and then generalize the properties to n stages. Assuming m training vectors of length p and NLT1 in Fig. 2.1 to be the identity operator (X(n)=X'(n)), we define

$$X = \begin{bmatrix} x_1^t \\ x_2^t \\ \cdot \\ \cdot \\ \cdot \\ x_m^t \end{bmatrix},$$

$$Y = \begin{bmatrix} y_1^t \\ y_2^t \\ \cdot \\ \cdot \\ \cdot \\ y_m^t \end{bmatrix},$$

$$D_1^1 = \begin{bmatrix} d(1) & d(2) & \cdots & d(m) \end{bmatrix}^t,$$

$$W_1 = \begin{bmatrix} a_1 & a_2 & \cdots & a_p \end{bmatrix}^t,$$

$$W_2 = \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix}^t.$$

X and Y are m $\times$ p matrices. Each row of X or Y represents an input vector of SNNl or SNNZ, respectively. D_1^1 is the desired output vector of length m. Using the delta rule to train SNNl corresponds ideally to finding the least-squares solution for $XW_1=D_1^1$. The output of SNNl is o_1^1 which can be expressed as [DeEr91]

$$o_1^1=XX^+D_1^1=AD_1^1, \tag{4.1}$$

where X^+ is the generalized inverse of X, and the projection operator A is XX^+, which is positive semidefinite.

The error vector of SNNl is

$$e_1^1=D_1^1-o_1^1=(I-A)D_1^1. \tag{4.2}$$

We use e_1^1 as the desired output for SNNZ, to be also trained by the delta rule. The output of SNNZ after training can be expressed as

$$o_2^1=YY^+e_1^1=Be_1^1, \tag{4.3}$$

where we define $YY^+ \triangleq B$, which is also positive and semidefinite. Then,

$$e_2^1=e_1^1-o_2^1=(I-B)e_1^1. \tag{4.4}$$

With two stages, $o_1^1+o_2^1$ is the output, and the system error e_f is

$$e_f=D_1^1-(o_1^1+o_2^1)=e_2^1. \tag{4.5}$$

The above results can be considered to be the first sweep in a number of sweeps of forward-backward training. In the second sweep, the desired vector for SNNl is set equal to

$$D_1^2=o_1^1+e_2^1. \tag{4.6}$$

The new output of SNN1 is

$$o_1^2=A(o_1^1+e_2^1)=o_1^1+Ae_2^1, \tag{4.7}$$

because A is the projection operator, o_1^1 is in the space spanned by A, and $Ao_1^1=o_1^1$.

The new error signal for SNN1 is

$$e_1^2=D_1^2-o_1^2=(I-A)e_2^1. \tag{4.8}$$

After a straightforward derivation, we get

$$e_1^2=D_1^1-(o_1^2+o_2^1). \tag{4.9}$$

If we terminate the training at this point, the system output is $o_1^2+o_2^1$. Therefore e_1^2 is just the error of the system. If we continue to train SNN2, the new desired signal for SNN2 is

$$D_2^2=o_2^1+e_1^2. \tag{4.10}$$

The output of SNN2 becomes

$$o_2^2=BD_2^2=o_2^1+Be_1^2, \tag{4.11}$$

since o_2^1 is in the space spanned by B.

The error vector for SNN2, is

$$e_2^2=D_2^2-o_2^2=(I-B)e_1^2. \tag{4.12}$$

Using the same derivation leading to Eq.(9), we get

$$e_2^2=D_1^1-(o_1^2+o_2^2), \tag{4.13}$$

where e_2^2 is the error signal of the system at the end of the second sweep.

At the nth sweep, the desired output signal for SNNl is

$$D_1^n = o_1^{n-1} + e_2^{n-1}. \tag{4.14}$$

After training, the output of SNNl is

$$o_1^n = AD_1^n = o_1^{n-1} + Ae_2^{n-1}. \tag{4.15}$$

The error vector is

$$e_1^n = D_1^n - o_1^n = (I-A)e_2^{n-1}. \tag{4.16}$$

The error vector can also be written as

$$e_1^n = D_1^1 - (o_1^n + o_2^{n-1}). \tag{4.17}$$

At the nth sweep, the desired signal for **SNN2** is

$$D_2^n = o_2^{n-1} + e_1^n. \tag{4.18}$$

The output is

$$of = BD_2^n = o_2^{n-1} + Be_1^n. \tag{4.19}$$

The error is

$$e_2^n = D_2^n - o_2^n = (I-B)e_1^n. \tag{4.20}$$

Again, we note that

$$e_2^n = D_1^1 - (of + o_2^n), \tag{4.21}$$

where ef is the system error after the nth sweep.

From Eq. (4.2) and Eq. **(4.4)**, we get

$$||e_1^1||^2 = (D_1^1)^t(I-A)(D_1^1), \tag{4.22}$$

$$||e_2^1||^2 = (e_1^1)^t(I-B)(e_1^1) \leq ||e_1^1||^2. \tag{4.23}$$

Frorn Eq.(4.8) and Eq.(4.12), we get

$$||e_1^2||^2=(e_2^1)^t(I-A)(e_2^1)\leq||e_2^1||^2, \tag{4.24}$$

$$||e_2^2||^2=(e_1^2)^t(I-B)(e_1^2)\leq||e_1^2||^2. \tag{4.25}$$

From Eq.(4.16) and Eq.(4.20). we conclude that

$$||e_1^n||^2=(e_2^{n-1})^t(I-A)(e_2^{n-1})\leq||e_2^{n-1}||^2, \tag{4.26}$$

$$||e_2^n||^2=(e_1^n)^t(I-B)(e_1^n)\leq||e_1^n||^2. \tag{4.27}$$

Therefore,

$$||e_2^n||^2\leq||e_1^n||^2\leq||e_2^{n-1}||^2\leq . \cdot . \leq||e_1^2||^2\leq||e_2^1||^2\leq||e_1^1||^2. \tag{4.28}$$

We will see in the next section that

$$\lim_{n\to\infty}||e_1^n||^2=||e||^2, \tag{4.29}$$

$$\lim_{n\to\infty}||e_2^n||^2=||e||^2\ , \tag{4.30}$$

where $||e||^2$ is the square error sum of the function-link network which has the same input NLT's as used in the PSHNN.

4.3. Asymptotic Properties of a Two-Stage PSHNN with Forward-Backward Training

Consider a function-link network as shown in Fig. 4.1. Let X denote an input vector, Y be a nonlinear transformation of X and D be the desired output vector. X and Y are m×n matrices, D is an m×1 vector, and W is a 2n×1 weight matrix.

Using the delta rule to train W corresponds approximately to finding the leastsquares solution for

$$(X,Y)W=D,$$

where (X,Y) denotes the concatenation of X and Y. The leastsquares solution is

$$\overline{W}=(X,Y)^{+}D,$$

where $(X,Y)^{+}$ is the pseudo-inverse of (X,Y).
The output vector is

$$o=(X,Y)(X,Y)^{+}D,$$

Therefore, the error vector is

$$e=D-o=(I-(X,Y)(X,Y)^{+})D. \tag{4.31}$$

If we use PSHNN with forward-backward training, Eqs. (4.2), (4.4), (4.8), (4.13) and $D_1^1 = D$ in this case lead to

$$e_1^1=(I-XX^{+})D, \tag{4.32}$$

$$e_2^1=(I-YY^{+})(I-XX^{+})D, \tag{4.33}$$

$$e_1^2=(I-XX^{+})[(I-YY^{+})(I-XX^{+})]D, \tag{4.34}$$

$$e_2^2=[(I-YY^{+})(I-XX^{+})]^2D, \tag{4.35}$$

.

.

$$e_1^n=(I-XX^+)[(I-YY^+)(I-XX^+)]^{n-1}D, \tag{4.36}$$

$$e_2^n=[(I-YY^+)(I-XX^+)]^nD, \tag{4.37}$$

$$e_1^{n+1}=(I-XX^+)[(I-YY^+)(I-XX^+)]^nD. \tag{4.38}$$

We will need the following properties to prove the main theorem of this section:

Property 1: The null space $N(XX^t+YY^t)$ is equivalent to the intersection of the null space $N(XX^t)$ and the null space $N(YY^t)$.

Proof:

(i) For any vector $y \in N(XX^t) \cap N(YY^t)$

it is obvious that $y \in N(XX^t+YY^t)$.

(ii) For any vector $y \in N(XX^t+YY^t)$

$(XX^t+YY^t)y=0$

$\Rightarrow$ $XX^ty=-YY^ty$

Therefore, $y^tXX^ty=-y^tYY^ty$

Since XX^t and YY^t are positive semidefinite

$y \in N(XX^t)$ and $y \in N(YY^t)$ □

In addition, the following properties are needed:

Property 2: The projection operators $P_{N(XX^t)}$ and $P_{N(YY^t)}$ satisfy

$$\lim_{n\to\infty}(P_{N(XX^t)}P_{N(YY^t)})^n = P_{N(XX^t)\cap N(YY^t)}, \tag{4.39}$$

which can be found in Nakano [Naka53]. This property tells us that the projection not in the intersection of $N(XX^t)$ and $N(YY^t)$ will gradually vanish as n goes to infinity. The projection in the intersection of $N(XX^t)$ and $N(YY^t)$ will be preserved.

Property 3:

$$P_{N(XX^t)}P_{N(XX^t)\cap N(YY^t)} = P_{N(XX^t)\cap N(YY^t)}, \tag{4.40}$$

which can be found in **Hartwig and Drazin** [HaDr82] and Nakano [Naka53].

Next, we will state and prove the main theorem:

Theorem 1:

$$\lim_{n\to\infty} e_1^{n+1} = \lim_{n\to\infty} e_1^n = e, \tag{4.41}$$

$$\lim_{n\to\infty} e_2^n = e. \tag{4.42}$$

Proof:

The projection matrices are

$$(I - XX^+) \triangleq P_{N(XX^t)},$$

$$(I-YY^+)\triangleq P_{N(YY^t)}.$$

Cornparing Eqs. (4.31), (4.37) and (4.38), sufficient conditions for Eq. (4.41) and Eq. (4.42) to hold are

$$\lim_{n\to\infty}(I-XX^+)[(I-YY^+)(I-XX^+)]^n=[I-(X,Y)(X,Y)^+], \tag{4.43}$$

$$\lim_{n\to\infty}[(I-YY^+)(I-XX^+)]^n=[I-(X,Y)(X,Y)^+]. \tag{4.44}$$

Using the projection operators, we get

$$[(I-YY^+)(I-XX^+)]^n=(P_{N(YY^t)}P_{N(XX^t)})^n. \tag{4.45}$$

From Property 1, we have

$$N(XX^t)\cap N(YY^t)=N(XX^t+YY^t)=N((X,Y)(X,Y)^t).$$

Therefore,

$$P_{N(XX^t)\cap N(YY^t)}=P_{N((X,Y)(X,Y)^t)}. \tag{4.46}$$

We know that

$$P_{N((X,Y)(X,Y)^t)}=[I-(X,Y)(X,Y)^+] \tag{4.47}$$

From Eqs. (4.39), (4.45), (4.46) and (4.47), we conclude that

$$\lim_{n\to\infty}[(I-YY^+)(I-XX^+)]^n=[I-(X,Y)(X,Y)^+].$$

Eq. (4.44) to be proved follows directly from Property 3:

$$\lim_{n\to\infty}(I-XX^+)[(I-YY^+)(I-XX^+)]^n=[I-(X,Y)(X,Y)^+] \qquad \square$$

The theorem proved above means that, as n grows larger, the error vectors e_1^n and e_2^n approach the error vector e for the pseudoinverse solution if a single total network was built without stages with the total input vector.

4.4. Asymptotic Properties for an N-Stage Network

When the number of stages is 2, forward-backward training is the same as circular training discussed in Ref. [DeEr91]. In the circular training algorithm with n stages, after training SNN(n), we train SNN(1). In forward-backward training, we will train SNN(n-1) after training SNN(n), followed by SNN(n-2) and so on. From the first stage to the last stage, we have a forward path training, and then from the last stage to the first stage, we have a backward path training. One sweep training consists of a forward path and a backward path training. We will call this training procedure the forward-backward traing algorithm.

For the sake of brevity , we will discuss the 3-stage PSHNN. All the properties of the 3-stage network can be derived for the n-stage network in the same way. Referring to Fig. 2.1 and supposing $X=X'$, we define $N[XX^t]=A$, $N[YY^t]=B$, and $N[ZZ^t]=C$ to represent the null space of (XX^t), (YY^t) and (ZZ^t), respectively. After the first stage is trained, the error vector is

$$e_{1f}^1=[P_A]D, \tag{4.48}$$

where P_A is the projection matrix of A, and D is the desired output vector. The superscript of the error vector denotes the number of sweeps, the Arabic number on the subscript denotes the number of stages, and the letter "f" on the subscript means forward path training. Following the same procedure as in Section 4.3, we have

$$e^1_{2f}=[P_BP_A]D, \tag{4.49}$$

$$e^1_{3f}=[P_CP_BP_A]D. \tag{4.50}$$

After training three stages in the forward path, we transmit the error of the third stage to the second stage and modify the desired output of the second stage in order to train the second stage, and get the error vector

$$e^1_{2b}=[P_BP_CP_BP_A]D, \tag{4.51}$$

where the letter "b" in the subscript means backward training path. After training the second stage, we train the first stage and get the error vector

$$e^1_{1b}=[P_AP_BP_CP_BP_A]D. \tag{4.52}$$

Now, the first sweep is over, and the second sweep starts.

Following the same procedure as above, we get the following error vectors in the second sweep:

$$\begin{aligned} e^2_{1f}&=P_A[P_AP_BP_CP_BP_A]D \\ &=[P_AP_BP_AP_BP_A]D \\ &=e^1_{1b}, \end{aligned} \tag{4.53}$$

$$e^2_{2f}=P_BP_A[P_AP_BP_CP_BP_A]D, \tag{4.54}$$

$$e^2_{3f}=P_CP_BP_A[P_AP_BP_CP_BP_A]D, \tag{4.55}$$

$$e^2_{2b}=P_BP_CP_BP_A[P_AP_BP_CP_BP_A]D, \tag{4.56}$$

$$e_{1b}^{2}=[P_A P_B P_C P_B P_A]^2 D$$

$$=e_{1f}^{3}. \qquad (4.57)$$

After the nth sweep training, the error vector of the first stage becomes

$$e_{1b}^{n}=e_{1f}^{n+1}=[P_A P_B P_C P_B P_A]^n D. \qquad (4.58)$$

Similar to the derivation of Eq. (4.31), the error vector for a 3-stage function-link network is

$$e=[I-(X,Y,Z)(X,Y,Z)^{+}]D$$

$$=[P_{N(XX^t+YY^t+ZZ^t)}]D, \qquad (4.59)$$

where $N(XX^t+YY^t+ZZ^t)$ denotes the null space of $(XX^t+YY^t+ZZ^t)$.

We also need the following properties:

Property 1.a: The null space $N(XX^t+YY^t+ZZ^t)$ is equivalent to the intersection of the null space $N(XX^t)$, the null space $N(YY^t)$ and the null space $N(ZZ^t)$.

Proof:

(i) For any vector $a \in N(XX^t) \cap N(YY^t) \cap N(ZZ^t)$,

it is obvious that $a \in N(XX^t+YY^t+ZZ^t)$.

(ii) For any vector $a \in N(XX^t+YY^t+ZZ^t)$,

then $(XX^t+YY^t+ZZ^t)a=0$.

Therefore, $a^t(XX^t+YY^t+ZZ^t)a=0$,

$\Rightarrow$ $a^tXX^ta+a^tYY^ta+aZZ^ta=0$.

Because (XX^t), (YY^t), and (ZZ^t) are positive semidefinite,

we have $a^tXX^ta=0$, $a^tYY^ta=0$ and $a^tZZ^ta=0$.

These imply $a \in N(XX^t)$, $a \in N(YY^t)$, and $a \in N(ZZ^t)$. $\square$

Property 2.a:

$$\lim_{n \to \infty} (P_A P_B P_C P_B P_A)^n = P_{A \cap B \cap C} \tag{4.60}$$

which was proved by Pyle [Pyle67].

From Eq. (4.59) and property 1.a, we get

$$e = (P_{N(XX^t + YY^t + ZZ^t)})D = (P_{A \cap B \cap C})D. \tag{4.61}$$

By using Property 2.a, Eq. (4.58) and Eq. (4.61), we obtain the main theorem of this section:

Theorem 2:

$$\lim_{n \to \infty} e_{1b}^n = e. \tag{4.62}$$

Since Property 2.a still holds for the intersection of n projection matrices, the generalization of Theorem 2 to the n-stage PSHNN with forward-backward training is obvious.

The results of Theorem 1 of Sec. 4.3 is based on the two-stage PSHNN. For the two-stage PSHNN, circular training is the same as the forward-backward training. An interesting question is whether circular training gives the same results as forward-backward training for the n-stage networks. This is conjectured to be true since many experiments show that [Pyle67]

$$\lim_{n \to \infty} (P_C P_B P_A)^n = P_{A \cap B \cap C}. \tag{4.63}$$

Experimentally, we have also observed that circular training gives the same results as forward-backward training.

4.5. Asymptotic Properties for the Suboptimal Solutions

In Sec. 4.4, we discussed the asymptotic property of PSHNN with forward-backward training when each stage gives the exact least-squares solution. In this section, we generalize the asymptotic property to the suboptimal least-squares solution due to the use of the delta rule. We discuss the case of the two-stage PSHNN, and the results can be easily extended to the n-stage PSHNN.

Assuming a two-stage network, the square error sum $||\mathbf{e}_2^1||^2$ in Eq. (4.23) is based on the optimal least-squares solution for the second stage. The least-squares error vector $\mathbf{e}_2^1$ is in the null space of $[YY^t]$. Defining $\xi_{ls} \triangleq ||\mathbf{e}_2^1||^2$, Eq. (4.23) can be written as

$$\xi_{ls}=||(I-YY^+)\mathbf{e}_1^1||^2=||P_{N(YY^t)}\mathbf{e}_1^1||^2, \tag{4.64}$$

where $P_{N(YY^t)}$ is the projection matrix to the null space of YY^t.

In reality, the square error sum we get by using the delta rule is based on a suboptimal least-squares solution. The suboptimal square error sum denoted as $\hat{\xi}_{ls}$ can be expressed as [Alex86], [Hayk91]

$$\hat{\xi}_{ls}=m(\xi_{min}+\xi_{exc}), \tag{4.65}$$

where m denotes the number of input vectors. ξ_{min} is the minimum mean square error (MSE) by solving the normal equation

$$E[Y_N(n)Y_N(n)^t]W_N=E[e_1^1(n)Y_N(n)], \tag{4.66}$$

where $Y_N(n)=[y(n),y(n-1),\cdots,y(n-N+1)]^t$, and N denotes the number of weights of SNN2 of Fig. 2.1; ξ_{exc} is due to the actual LMS weights jitter, and is sometimes referred to as the excess MSE. If we assume the sequence $y(n)$ is stationary and ergodic, then $m\xi_{min}$ in Eq. (4.65) gradually approaches the optimal square error sum ξ_{ls} as m grows. Thus, approximating $m\xi_{min}$ by ξ_{ls}, Eq. (4.65) can be written as

$$\hat{\xi}_{ls}=\xi_{ls}+m\xi_{exc} \ . \tag{4.67}$$

ξ_{exc} is proportional to gain η used in training. Choosing smaller η achieves better suboptimal square error sum $\hat{\xi}_{ls}$, but then the learning rate is slower. So, there is a trade-off involved in choosing the value of η.

We show below that the error reduction properties derived in Sec. 4.2 still hold in practise with the square error sum $\hat{\xi}_{ls}$ based on a suboptimal least-squares solution.

For the sake of brevity, we consider a two-stage PSHNN with NLT1 being the identity operator. D_1^1 is the desired vector for the first stage network in the first sweep. The output vector of the first stage based on the optimal least squares solution is [HoKu71], [RaMi71]

$$o_1^1=P_{col[XX^t]}D_1^1 \ . \tag{4.68}$$

The output vector $\hat{o}_1^1$ based on the suboptimal leastsquares solutions W'_1 is written as

$$\hat{o}_1^1=XW'_1 \ . \tag{4.69}$$

This shows that $\hat{o}_1^1 \in col[XX^t]$. $\hat{o}_1^1$ can be written as

$$\hat{o}_1^1=P_{col[XX^t]}D_1^1+b_1^1 \ , \tag{4.70}$$

where the vector b_1^1 also belongs to the column space of $[XX^t]$. This is graphically shown in Fig. 4.2. The magnitude of b_1^1 can be written as

$$||b_1^1||=c_1^1||P_{col[XX^t]}D_1^1|| \ , \tag{4.71}$$

where c_1^1 satisfies $0<c_1^1<1$ in practise. Thus the error vector of SNN1 in the first sweep is

$$\hat{e}_1^1 = P_{N[XX^t]} D_1^1 - b_1^1 \ . \tag{4.72}$$

$\hat{e}_1^1$ is also the desired vector for the second stage network in the first sweep. Referring to Fig. 4.3, and using the same procedure as above, we get the suboptimal output vector $\hat{o}_2^1$ of SNN2 in the first sweep as

$$\hat{o}_2^1 = P_{col[YY^t]} \hat{e}_1^1 + b_2^1 \ , \tag{4.73}$$

where the vector b_2^1 belongs to the column space of $[YY^t]$, and the magnitude of b_2^1 is

$$||b_2^1|| = c_2^1 ||P_{col[YY^t]} \hat{e}_1^1|| \ , \tag{4.74}$$

where c_2^1 also satisfies $0<c_2^1<1$ in practise. The error vector of SNN2 in the first sweep is

$$\hat{e}_2^1 = P_{N[YY^t]} \hat{e}_1^1 - b_2^1 \ . \tag{4.75}$$

Since $P_{N[YY^t]}\hat{e}_1^1$ and b_2^1 are orthogonal to each other, we get

$$\begin{aligned} ||\hat{e}_2^1||^2 &= ||P_{N[YY^t]}\hat{e}_1^1||^2 + ||b_2^1||^2 \\ &\leq ||P_{N[YY^t]}\hat{e}_1^1||^2 + ||P_{col[YY^t]}\hat{e}_1^1||^2 = ||\hat{e}_1^1||^2 \ . \end{aligned} \tag{4.76}$$

Thus, $||\hat{e}_2^1||^2$ is less than $||\hat{e}_1^1||^2$ as long as c_2^1 is less than 1, which is definitely true in practise.

On the second sweep, the desired vector of SNN1 is $\hat{e}_2^1 + \hat{o}_1^1$. Following the same procedure as above, the suboptimal output vector $\hat{o}_1^2$ of SNN1 in the second sweep is found as

$$\begin{aligned} \hat{o}_1^2 &= P_{col[XX^t]}(\hat{e}_2^1 + \hat{o}_1^1) + b_1^2 \\ &= \hat{o}_1^1 + P_{col[XX^t]}\hat{e}_2^1 + b_1^2 \ , \end{aligned} \tag{4.77}$$

and

$$||b_1^2||=c_1^2||P_{col[XX^t]}(\hat{e}_1^2+\hat{o}_1^1)|| , \qquad (4.78)$$

where $\hat{o}_1^1\in col[XX^t]$, $b_1^2\in col[XX^t]$ and $0<c_1^2<1$. The error vector $\hat{e}_1^2$ of SNN1 in the second sweep is

$$\hat{e}_1^2=(\hat{e}_2^1+\hat{o}_1^1)-\hat{o}_1^2$$

$$=P_{N[XX^t]}\hat{e}_2^1-b_1^2 . \qquad (4.79)$$

The desired vector of SNN2 in the second sweep is $\hat{e}_1^2+\hat{o}_2^1$. The suboptimal output vector $\hat{o}_2^2$ of SNN2 in the second sweep is

$$\hat{o}_2^2=P_{col[YY^t]}(\hat{e}_1^2+\hat{o}_2^1)+b_2^2$$

$$=\hat{o}_2^1+P_{col[YY^t]}\hat{e}_1^2+b_2^2 , \qquad (4.80)$$

and

$$||b_2^2||=c_2^2||P_{col[YY^t]}(\hat{e}_1^2+\hat{o}_2^1)|| , \qquad (4.81)$$

where $\hat{o}_2^1\in col[YY^t]$, $b_2^2\in col[YY^t]$, and $0<c_2^2<1$. The error vector $\hat{e}_2^2$ of SNN2 in the second sweep is

$$\hat{e}_2^2=(\hat{e}_1^2+\hat{o}_2^1)-\hat{o}_2^2$$

$$=P_{N[YY^t]}\hat{e}_1^2-b_2^2 . \qquad (4.82)$$

Using Eq. (4.72) and Eq. (4.75), and letting $A\triangleq N[XX^t], B\triangleq N[YY^t]$; the suboptimal error vector $\hat{e}_1^1$ of the first stage in the first sweep becomes

$$\hat{e}_1^1=P_AD_1^1-b_1^1 . \qquad (4.83)$$

The suboptimal error vector $\hat{e}_2^1$ of the second stage in the first sweep becomes

$$\hat{e}_2^1=P_B\hat{e}_1^1-b_2^1$$

$$=P_BP_AD_1^1-P_Bb_1^1-b_2^1 . \qquad (4.84)$$

Using **Eq.** (4.79) and **Eq.** (4.84), the suboptimal error vector $\hat{e}_1^2$ of the first stage in the second sweep becomes

$$\hat{e}_1^2=(P_AP_B)P_AD_1^1-P_AP_Bb_1^1-P_Ab_2^1-b_1^2\ ,\tag{4.85}$$

where $b_1^2\in col[XX^t]$. The suboptimal error vector $\hat{e}_2^2$ of the second stage in the second sweep becomes

$$\hat{e}_2^2=(P_BP_A)^2D_1^1-(P_BP_A)P_Bb_1^1-(P_BP_A)b_2^1-P_Bb_1^2-b_2^2\ ,\tag{4.86}$$

where $b_2^2\in col[YY^t]$.

Following the same procedure, the suboptimal error vector $\hat{e}_1^n$ of the first stage in the nth sweep becomes

$$\hat{e}_1^n=(P_AP_B)^{n-1}P_AD_1^1-\sum_{k=1}^{n}(P_AP_B)^{n-k}b_1^k-\sum_{k=1}^{n-1}(P_AP_B)^{n-1-k}P_Ab_2^k\ .\tag{4.87}$$

The suboptimal error vector $\hat{e}_2^n$ of the second stage in the nth sweep becomes

$$\hat{e}_2^n=(P_BP_A)^nD_1^1-\sum_{k=1}^{n}(P_BP_A)^{n-k}P_Bb_1^k-\sum_{k=1}^{n}(P_BP_A)^{n-k}b_2^k\ ,\tag{4.88}$$

where $b_1^i\in col[XX^t]$, and $b_2^i\in col[YY^t]$ for any positive integer i. Since the directions of b_1^i and b_2^i are random, the magnitudes of the summation terms in **Eq.** (4.87) and **Eq.** (4.88) are **small** in the mean sense. Therefore, the first term on the right hand side of **Eq.** (4.87) or **Eq.** (4.88) can be considered as the dominant term in real-world applications. Then, the error reduction property of **Eq.** (4.28) in **Sec.** 4.2 still holds for this suboptimal case.

In practise, if n is large enough such that $(P_BP_A)^n=P_{A\cap B}$, and $m>n$, we can rewrite **Eq.** (87) and **Eq.** (88) as follows:

$$\hat{e}_1^m=e-\sum_{k=m-n+1}^{m}(P_AP_B)^{m-k}b_1^k-\sum_{k=m-n+1}^{m-1}(P_AP_B)^{m-1-k}P_Ab_2^k\ ,\tag{4.89}$$

and

$$\bar{e}_2^m = e - {}_{k=m-n+1} (P_B P_A)^{m-k} P_B b_1^k - \sum_{k=m-n+1}^{m} (P_B P_A)^{m-k} b_2^k , \qquad (4.90)$$

The error vector e in Eq. (4.89) and Eq. (4.90) is the vector in Eq. (4.31), which is the optimal least-squares error vector of the function-link network as shown in Fig.. 4.1. We also see that no matter how big m is, there are at most n vectors in each summation term of Eq. (4.89) and Eq. (4.90).

4.6. Experimental Results

The theoretical results discussed above were tested with a speech signal sampled at 10 khz. 100 Samples were used to train the network by the delta rule. The gain factor we used in the experiments was 0.001. No momentum term was used. The input pointwise nonlinear transformations used in the experiments are the following:

(A) SIGMOID 1 (Sig. I) :$(0<y<1)$

$$y = \frac{1}{1+e^{-x}}$$

(B) SIGMOID 2 (Sig. II) : $(-1<y<1)$

$$y = 2 \times \text{sigmoid}(x) - 1$$

(C) THRESHOLD 1 (Th. I):

$$y = 1 \quad \text{if } x \geq 0$$

$$y = 0 \quad \text{if } x < 0$$

(D) THRESHOLD 2 (Th. II):

$$y = 1 \quad \text{if } x \geq 0$$

$$y = -1 \quad \text{if } x < 0$$

(E) SQUARE :

$$y = x^2$$

In the experiments, we first normalized the input data in the range $\{-1,1\}$. Five weights were used for each stage of a two-stage PSHNN. Ten weights were used for the function-link network. The initial matrix of the network was set equal to the covariance matrix of the input data.

Table 4.1 are the results of the function-link network with the ten weights listed as a function of the five types of NLT's.

Tables 4.2 thru 4.6 are the results of the two-stage PSHNN with forward-backward training. Table 4.2 is for Sig.I, Table 4.3 for Sig.II, Table 4.4 for Th.I, Table 4.5 for Th.II, and Table 4.6 for the square NLT.

Tables 4.2 and 4.3 for Sig.I and Sig.II cases show that the PSHNN with forward-backward training has more error reduction and faster convergence rate than the function-link network. With Th.II and square NLT's, the PSHNN and the function-link network are about the same both in error reduction and convergence rate. With Sig.II NLT, there is negligible error reduction both in the PSHNN and the function-link network. This is because the input data was normalized in the range $\{-1,1\}$, and this causes x and y to be almost the same in this range.

Tables 4.7 and Table 4.8 are the results of the function-link network with three-stage input vectors of length 5 concatenated as a total input vector to the network. Tables 4.9 thru 4.11 show the error reduction performance of the corresponding three-stage PSHNN with forward-backward training, In the first stage, 100 iterations were used during the first sweep, and 300 iterations were used during the succeeding sweeps. The number of iterations of the second and the third stages were 500, and 900, respectively. In Tables 4.9, 4.10 and 4.11, the notations used mean err1f $= ||e^i_{1f}||^2$, err2f $= ||e^i_{2f}||^2$, err3f $= ||e^i_{3f}||^2$, and err2b $= ||e^i_{2b}||^2$. The superscript "i" denotes the number of sweeps as in Section 4.2. From Tables 4.7 and 4.8, we see that the convergence rate is rather slow for the function-link networks. Comparing Tables 4.7 and 4.8 to Tables 4.9, 4.10 and 4.11, we observe that PSHNN with forward-backward training is superior to the function-link network in terms of both convergence rate and error reduction.

Table 4.1. Performance of the Function-Link Network in Speech Prediction ($err = ||e||^2$).

type of NLT	err	number of iterations
Sig.I	2.1344	1000
Th.II	2.027	1000
Sig.II	2.1291	1000
Th.I	2.0459	600
Sqre.	1.8862	1000

Table 4.2. Performance of PSHNN with NLT Sig.I in Speech Prediction $(err1=||e_1^i||^2, err2=||e_2^i||^2)$.

n-th sweep	err1	err2	# of iterations	
			stage1	stage2
n=1	2.1353	1.9336	100	1000
n=2	1.8718	1.8524	900	100
n=3	1.8460	1.8416	900	100

Table 4.3. Performance of PSHNN with NLT Sig.II in Speech Prediction ($err1=||e_1^i||^2, err2=||e_2^i||^2$).

n-th sweep	err1	err2	# of iterations	
			stage1	stage2
n=1	2.1353	2.1390	100	1000
n=2	2.1343	2.1385	900	100
n=3	2.1336	-	900	-

Table 4.4. Performance of PSHNN with NLT Th.I in Speech Prediction ($err1=||e_1^i||^2, err2=||e_2^i||^2$).

n-th sweep	err1	err2	# of iterations	
			stage1	stage2
n=1	2.1352	2.0925	100	200
n=2	2.0699	2.0585	900	200
n=3	2.0514	2.0481	900	200
n=4	2.0457	2.0448	900	200

Table 4.5. Performance of PSHNN with NLT Th.II in Speech Prediction ($err1=||e_1^i||^2, err2=||e_2^i||^2$).

n-th sweep	err1	err2	# of iterations	
			stage1	stage2
n=1	2.1353	2.0282	100	100
n=2	2.0312	2.0250	500	100
n=3	2.0034	-	600	-

Table 4.6. Performance of PSHNN with NLT Square in Speech Prediction ($err1=||e_1^i||^2, err2=||e_2^i||^2$).

n-th sweep	err1	err2	# of iterations	
			stage1	stage2
n=1	2.1353	1.9326	100	600
n=2	1.8973	1.8896	900	600
n=3	1.8872	1.8867	900	600
n=4	1.8864	1.8863	900	600

Table 4.7. 3-Stage Function-Link Network as a Function of Input Nonlinearity with 900 Iterations (err= $||e||^2$).

Type of NLT		err
Stage II	Stage III	
Sig.I	Th.II	2.0167
Th.I	Sig.I	1.9980
Square	Sig.I	1.8818

Table 4.8. 3-Stage Function-Link Network as a Function of Input Nonlinearity with 2900 Iterations (err= $||e||^2$).

Type of NLT		err
Stage II	Stage III	
Sig.I	Th.II	2.0149
Th.I	Sig.I	1.9906
Square	Sig.I	1.8811

Table 4.9. Performance of PSHNN with NLT1 Sig.I & NLT2 Th.II in Speech Prediction.

n-th Sweep Training	Square Error Sum			
	err1f	err2f	err3f	err2b
n=1	2.1353	1.9377	1.8393	1.8758
n=2	1.8122	1.7584	1.7543	-

Table 4.10. Performance of PSHNN with NLT1 Th.I & NLT2 Sig.I in Speech Prediction.

n-th Sweep Training	Square Error Sum			
	err1f	err2f	err3f	err2b
n=1	2.1353	2.0924	1.9210	1.8957
n=2	1.8750	1.8592	1.8264	-

Table 4.11. Performance of PSHNN with NLT1 Square & NLTB Sig.I in Speech Prediction.

n-th Sweep Training	Square Error Sum			
	err1f	err2f	err3f	err2b
n=1	2.1353	1.9330	1.6973	1.6812
n=2	1.6705	1.6631	1.6399	

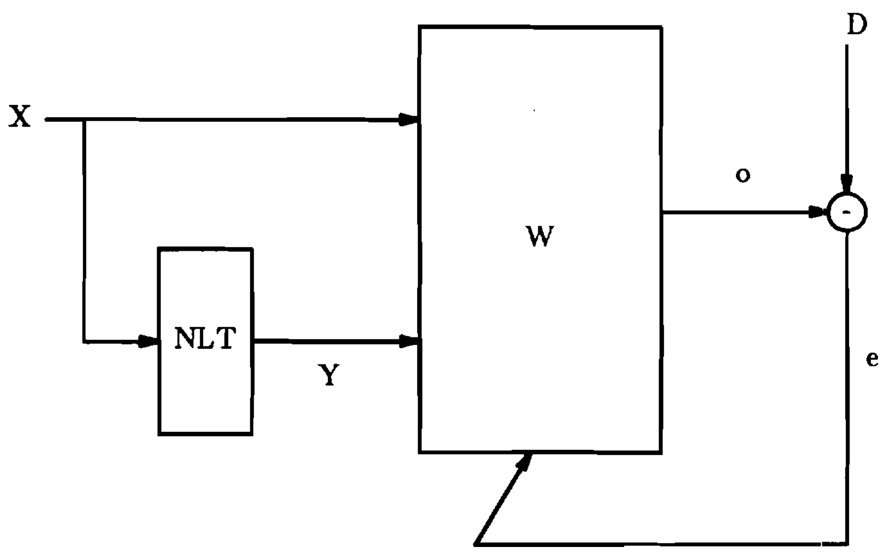

Figure 4.1. Block Diagram of a Function-Link Network.

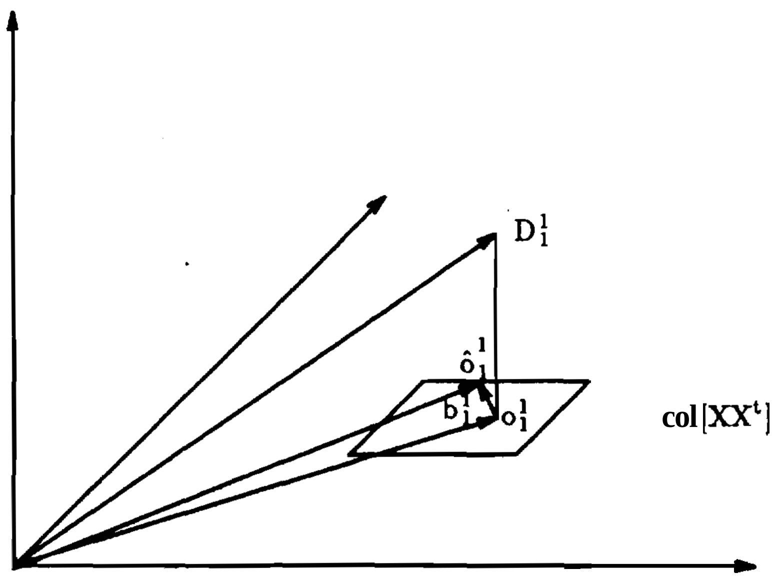

Figure 4.2. Graphical Representation of Suboptimal Solution for SNN1.

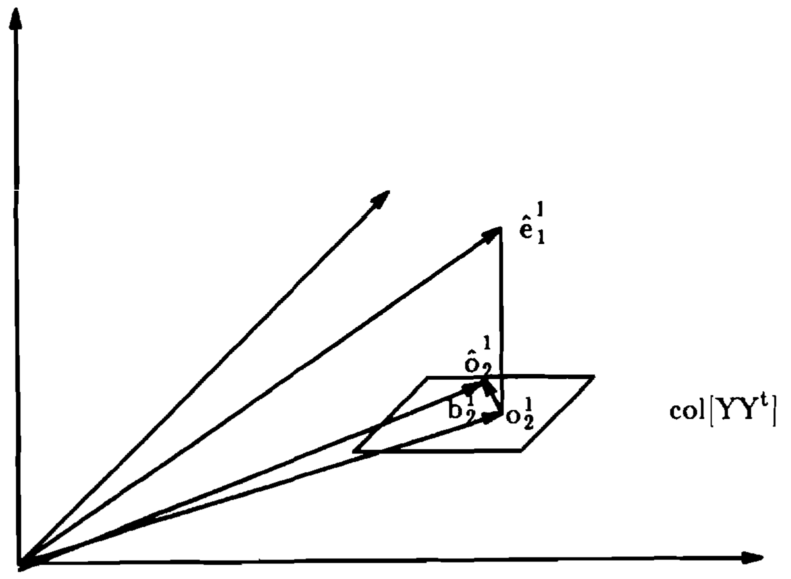

Figure 4.3. Graphical Representation of Suboptimal Solution for SNN2.

CHAPTER 5
LEARNING INPUT NONLINEAR TRANSFORMATIONS

5.1. Introduction

In Chapter 2, we discussed the generalieation of the **PSHNN's** with continuous input and output [**ErDe911**]. It was shown that stages are generated by nonlinearly transforming input vectors, and each new stage attempts to correct the errors of the previous stage. It is also shown that any input nonlinear transformation helps the system achieve smaller mean square error (MSE) than the MSE with linear prediction. By implementing the PSHNN stages in parallel, the speed of processing with several stages is the same as with one stage. The suboptimal error reduction property was also proved. An important research issue is how to minimiee the input **NLT's.** We proposed an effective approach called the revised backpropagation **(RBP)** network [**ErDe912**]. The RBP algorithm consists of two training steps, denoted as step I and step II, respectively. During step I, the **RBP** is the same as the usual backpropagation (BP) algorithm. During step **II,** we fix the weights between the input layer and the hidden layers, but retrain the weights between the last hidden and the output layers by the delta rule. In this chapter, the algorithm used during step II of the RBP is extended to incorporate the least mean absolute value (LMAV) criterion.

It was discussed in Chapter 4 that further error reduction can be achieved in a s n-stage PSHNN by forward-backward or circular training. The asymptotic properties show that the forward-backward training of n-stage PSHNN's until convergence is equivalent to the pseudo-inverse solution for a single total network designed in the least-squares sense to the total input vector consisting of the actual input vector and its additional nonlinear transformations [DeEr91], [DeEr921]. The error reduction property by forward-backward training stated above was based on the fixed input NLT of each stage of the PSHNN in every forward-backward sweep. In this chapter, we illustrate the technique which uses the BP algorithm with forward-backward training to learn the input NLT's of the PSHNN. In this case, the interconnection weights between the input and the hidden layers are allowed to change sweep by sweep. This means the PSHNN has different input NLT at each stage sweep by sweep. In this chapter, we also show the reason why the error reduction property still holds for this technique.

The chapter consists of 5 sections. In Sec. 5.2, we illustrate the method which uses the LMAV algorithm during step II of RBP. In Sec. 5.3, we show the reason why error reduction property of PSHNN which has BP stages with forward-backward training still holds. The experimental results of nonlinear speech prediction are given in Sec. 5.4. Simulations on nonlinear prediction of chaotic time series are discussed in Sec. 5.5.

6.2. RBP with the LMAV Algorithm

The RBP network consists of linear input and output units and nonlinear hidden units. One hidden layer is often sufficient [Miya88]. The hidden layers represent the nonlinear transformation of the input vector. The output of the jth unit of the kth layer is of the form

$$O_k(j)=f(\sum_{i=1}^{N_{k-1}} W_k(j,i)O_{k-1}(i)) ,$$

where N_{k-1} is the number' of output nodes of the (k-1)th layer; O_{k-1} is the output vector of the (k-1)th layer; $W_k(.,.)$ are the weights connecting the (k-1)th and the kth layers, and f(.) is the nonlinear activation function, assumed to be differentiable and usually chosen monotone nondecreasing.

The RBP with the LMAV algorithm also consists of two training steps, denoted as step I and step **II**, respectively. During step I, the **RBP** is the same as the usual **backpropagation** (BP) algorithm [**Rume88**]. During step **II**, we **fix** the weights between the input layer and the hidden layers, but retrain the weights between the last hidden and the output layers by the LMAV rule.

The RBP network with the LMAV algorithm is shown in Fig. 5.1. Let **X(n)** be the input vector sequence; the output vector of the last hidden layer is **Y(n)** which can be considered as the result of nonlinear transformation of **X(n)**. W are weights between the last hidden and the output layers. The least mean absolute value (LMAV) rule for the weight vector W is [**Bell87**]

$$W(n+1)=W(n)+\eta Y(n+1) \text{ sign } e(n+1) , \tag{5.1}$$

$$e(n+1)=d(n+1)-Y^t(n+1)W(n) , \tag{5.2}$$

where sign e is +1 if e is positive, and -1 otherwise. The adaptation step factor η is a positive constant. We now want to study the convergence of LMAV rule by considering the'weight vector W as it moves toward the optimum W_*. Eq. (5.1) can be rewritten as

$$W(n+1)-W_*=W(n)-W_*+\eta Y(n+1)\,\text{sign}\,e(n+1)\,. \tag{5.3}$$

Taking the square error sum of both sides, we get

$$||W(n+1)-W_*||^2=||W(n)-W_*||^2+\eta^2||Y(n+1)||^2-2\eta|e(n+1)|$$
$$+2\eta\,\text{sign}\,e(n+1)[d(n+1)-Y^t(n+1)W_*]\,, \tag{5.4}$$

and

$$||W(n+1)-W_*||^2\leq||W(n)-W_*||^2+\eta^2||Y(n+1)||^2-2\eta|e(n+1)|$$
$$+2\eta|d(n+1)-Y^t(n+1)W_*|\,. \tag{5.5}$$

Let the length of W be N; taking the expectation of both sides yields

$$E(||W(n+1)-W_*||^2)\leq E(||W(n)-W_*||^2)+\eta^2N\sigma_y^2$$
$$-2\eta E(|e(n+1)|)+2\eta E_{\min}\,, \tag{5.6}$$

where the minimal error $E_{\min}$ is

$$E_{\min}=E(|d(n+1)-Y^t(n+1)W_*|)\,. \tag{5.7}$$

Convergence is obtained for any positive η, and the residual error E_R is bounded by [Bell87]

$$E_R\leq E_{\min}+\frac{\eta}{2}N\sigma_y^2\,, \tag{5.8}$$

where E_R is

$$E_R=\frac{1}{n+1}E(\sum_{p=1}^{n+1}|e(p)|)\,. \tag{5.9}$$

The advantage of RBP networks with the LMAV rule is that the LMAV rule is robust to outliers in a data set [MoTu87].

5.3. Error Reduction Property of PSHNN with BP Stages and Forward-Backward Training

Each stage of PSHNN can be any type of neural network. In this section, BP stages are utilized together with forward-backward training [DeEr921]. The BP stages are chosen as linear input and output units and a single hidden layer. The input vector is fed into all the BP stages in parallel as shown in Fig. 5.2. With a k-stage network, the first, the second, ... , the kth BP stage are trained in this order, followed by retraining of the (k-1)th, the (k-2)th, ... , the second BP stage. This constitutes one sweep. The interconnection weights between the input and the hidden layers are allowed to change sweep by sweep. Therefore, we generate a different input NLT in each sweep at every stage.

Referring to Fig. 5.2, X is the input vector and D_1^1 is the desired vector in the first sweep. After the first BP stage is trained, $\mathbf{Y_1}$ is the vector after input **NLT1** of X, and $\mathbf{o_1^1}$ is the output vector of the first stage in the first sweep. When the number of training iterations is sufficiently large, the weight vector between the hidden and the output layer will be near the least–squares solution. The simulation results in Table 3.3 also show this fact. Thus, we have approximately, [DeEr922]

$$\mathbf{o_1^1 = P_{col[Y_1 Y_1^t]} D_1^1} , \tag{5.10}$$

$$\mathbf{e_1^1 = P_{N[Y_1 Y_1^t]} D_1^1} , \tag{5.11}$$

where $\mathbf{P_{col[Y_1 Y_1^t]}}$ is the projection matrix to the column space of $\mathbf{[Y_1 Y_1^t]}$ and $\mathbf{P_{N[Y_1 Y_1^t]}}$ is the projection matrix to the null space of $\mathbf{[Y_1 Y_1^t]}$. After the second stage is trained, $\mathbf{Z_1}$ is the vector after input **NLT2** of X; $\mathbf{o_2^1}$ is the output vector of the second stage of the first sweep, and similarly,

$$\mathbf{o_2^1 = P_{col[Z_1 Z_1^t]} e_1^1} , \tag{5.12}$$

$$e_2^1 = P_{N[Z_1 Z_1^t]} e_1^1 \ . \tag{5.13}$$

Therefore, $||e_2^1||^2 \leq ||e_1^1||^2$.

In the second sweep, the desired vector for the first stage becomes $D_1^2 = o_1^1 + e_2^1$. A sufficient condition for further error reduction in the second sweep is that the BP network produces the vector Y_2 after input NLT1 of X in the second sweep such that $col[Y_1 Y_1^t] \subset col[Y_2 Y_2^t]$, or equivalently, $N[Y_2 Y_2^t] \subset N[Y_1 Y_1^t]$. In other words, the vector Y_2 is obtained by a better input NLT1 of X in the second sweep than that in the first sweep. All the experiments discussed in Sec. 5 always showed that further error reduction is achieved in the second sweep. Hence, we assume that the BP network has the ability to produce Y_2 satisfying the above sufficiency condition. Then, the output vector o_1^2 of the first stage in the second sweep is

$$\begin{aligned} o_1^2 &= P_{col[Y_2 Y_2^t]}(o_1^1 + e_2^1) \\ &= o_1^1 + P_{col[Y_2 Y_2^t]} e_2^1 \ , \end{aligned} \tag{5.14}$$

since $o_1^1 \in col[Y_1 Y_1^t]$ and $col[Y_1 Y_1^t] \subset col[Y_2 Y_2^t]$. The error vector e_1^2 of the first stage in the second sweep is

$$\begin{aligned} e_1^2 &= D_1^2 - o_1^2 \\ &= P_{N[Y_2 Y_2^t]} e_2^1 \ . \end{aligned} \tag{5.15}$$

Therefore, $||e_1^2||^2 \leq ||e_2^1||^2$.

The desired vector D_2^2 of the second stage in the second sweep is $e_1^2 + o_2^1$. The vector Z_2 is obtained after the input NLT2 of X in the second sweep. Under the same assumption discussed above, we have $col[Z_1 Z_1^t] \subset col[Z_2 Z_2^t]$, or eqaivalently, $N[Z_2 Z_2^t] \subset N[Z_1 Z_1^t]$. The output vector of of the second stage in the second sweep is

$$o_2^2 = P_{col[Z_2Z_2^t]}(e_1^2 + o_2^1)$$

$$= o_2^1 + P_{col[Z_2Z_2^t]}e_1^2 \; . \qquad (5.16)$$

The error vector e_2^2 of the second stage in the second sweep is

$$e_2^2 = D_2^2 - o_2^2$$

$$= P_{N[Z_2Z_2^t]}e_1^2 \; . \qquad (5.17)$$

Therefore, $||e_2^2||^2 \leq ||e_1^2||^2$.

Following the same procedure and under the same assumption, the vector Y_n is obtained after the input NLT1 of X in the nth sweep. The error vector e_1^n of the first stage in the nth sweep becomes

$$e_1^n = P_{N[Y_nY_n^t]}e_2^{n-1} \; , \qquad (5.18)$$

where $N[Y_nY_n^t] \subset N[Y_{n-1}Y_{n-1}^t] \subset \cdots \subset N[Y_2Y_2^t] \subset N[Y_1Y_1^t]$.

Therefore, $||e_1^n||^2 \leq ||e_2^{n-1}||^2$. The vector Z_n is obtained after the input NLT2 of X in the nth sweep, and the error vector e_2^n of the second stage in the nth sweep becomes

$$e_2^n = P_{N[Z_nZ_n^t]}e_1^n \; , \qquad (5.19)$$

where $N[Z_nZ_n^t] \subset N[Z_{n-1}Z_{n-1}^t] \subset \cdots \subset N[Z_2Z_2^t] \subset N[Z_1Z_1^t]$.

We conclude that

$$||e_2^n||^2 \leq ||e_1^n||^2 \leq ||e_2^{n-1}||^2 \leq \cdots \leq ||e_1^2||^2 \leq ||e_2^1||^2 \leq ||e_1^1||^2 \; . \qquad (5.20)$$

This result can be generalized to n-stage PSHNN's.

5.4. Experiments on Nonlinear Speech Prediction

The theoretical results discussed above were tested in the application of speech prediction. For this purpose, 100 speech samples at the sampling rate of 10 Khz were used to train and to test the network. In the experiments, we first normalized the data in the range [-1, 1.]. A sliding window of length 5 data points was used to predict the next signal value following the window.

Table 5.1 shows the performance in terms of the absolute error sum $||err||_1$ of a one stage network with the RBP stage and the LMAV rule, tabulated as a function of the training iterations of step I and step II. In this experiment, the gain factor $\eta=1.0$ was used during step I, and $\eta=0.01$ during step II; five input nodes and eight hidden nodes were used, resulting in 40 weights between the input and the hidden layers, and 8 weights between the hidden and the output layers. Thus, 48 weights need to be learned during step I, and only 8 weights need to be revised during step II. This indicates that the learning time of six iterations during step II is approximately the :learning time of one iteration during step I. We see from Table 5.1 that the absolute error sum $||err||_1=6.9461$ after 500 learning iterations in step I and 200 learning iterations in step II. The learning time of 500 iterations in step I and 200 iterations in step II for this one stage network with RBP and the LMAV rule is approximately the learning time of 534 iterations for the same network with the usual BP algorithm. The network with the usual BP algorithm achieved $||err||_1=7.1472$ after 650 iterations. In other words, the network with the RBP and the LMAV rule is observed to achieve a deeper minimum in absolute error sum by a shorter learning time than the network with the usual BP algorithm.

Next we discuss the experimental results when using PSHINN with BP stages and forward-backward training. Tables 5.2 thru 5.5 are the experiments on the PSHNN's with BP stages and forward-backward training as discussed in Sec. 5.3. The length of the input layer at each stage is five, and a gain factor of 0.5 is used throughout. Table 5.2 shows how error was reduced as a function of the number of iterations with a single BP network having 12 hidden units. The corresponding PSHNN's with the same number of interconnection weights were chosen as 3-stage, 3-stage and 4-stage networks in which each stage had 6, 4, and 3 hidden nodes respectively, and its training was based on backpropagation. Tables 5.3, 5.4 and 5.5 show how error was reduced stage by stage and sweep by sweep of forward-backward training. 1000 forward-backward sweeps of 2-stage network, 750 forward-backward sweeps of 3-stage network and 666 forward-backward sweeps of 4-stage network are equivalent to 50000 iterations of the previous single BP network since 50 iterations were used to train each stage of the PSHNN's. It is observed that the error reduction properties of the PSHNN's with two stages and three stages are better than those of the single BP network. The PSHNN's achieve the same error performance at about 600 sweeps with the 2-stage PSHNN and at 423 sweeps with the 3-stage PSHNN as the single BP network achieves with 50000 iterations. Both 2-stage and 3-stage PSHNN's had a reduction of learning time by about 40%. It also appears that both 2-stage and 3-stage PSHNN's converge towards a deeper minimum than the single stage BP network. However, the 4-stage PSHNN performed actually worse than the single BP network. Thus, there exists on optimal number of hidden nodes per stage for best performance. The 3-stage PSHNN performs best in terms of deeper minimum and faster convergence rate. More experiments with different sets of data are needed to substantiate this property. However, we think that this is the case since the same property was observed in other applications with systems having nonlinearities [AgEr91], [ErZB90].

5.5. Nonlinear Prediction of Chaotic Time Series

Chaotic systems arise in many physical situations such as onset of turbulence in fluids [RuTa71], [SwGo78], chemical reactions [ToKa79], lasers [Hake75], and plasma physics [RuHO80]. We selected two chaotic time series to test the RBP networks. The first chaotic time series was generated according to the classic logistic, or Feigenbaum map given by [Feig78], [LaFa87]

$$x(n)=x(n-1)[1.-x(n-1)] \text{ .}$$

In the following simulations, we used 100 data points generated by the chaotic system according to the equation above, and normalized the data in the range [0,1].

Tables 5.6 thru 5.8 are the simulation results with the RBP networks using the delta rule, tabulated as a function of the number of training iterations during step I. The number of hidden units are 2, 4 and 8, respectively. The number of training iterations was 200 during step II. The gain factor during step I was 0.1 in Tables 5.6 and 5.8, and was 1.0 in Table 5.8. The gain factor was 0.01 during step II. In Table 5.6, we see that the RBP network with 360 iterations during step I and 200 iterations during step II can reach the same square error sum by the usual BP network with 2000 training iterations. This means we need only 21% training time with the RBP network to achieve the same performance as with the usual BP network trained with 2000 iterations. In Table 5.7, after 120 iterations during step I and 200 iterations during step II, the RBP network reached the same performance as with the usual BP trained with 2000 iterations. Therefore, the training time of the RBP network is 10% of the training time of the usual BP network for the same performance. In Table 5.8, after 60 iterations and 200 iterations during step I and step II, respectively, the

RBP network achieved the same performance as the usual BP trained with **2000** iterations. In this case, the training time of the RBP network is 6% of the usual BP network.

Tables **5.9** thru **5.11** show the simulation results using the RBP networks with the LMAV rule, tabulated **as** a function of training iterations during step 1. The number of hidden units are **2, 4** and **8,** respectively. The number of training iterations was **100** during step **II.** The gain factor during step I was **1** in Tables **5.10** and **5.11,** and was **0.1** in Table **5.8.** The gain factor was **l.E-6** during step **II.** In Table **5.9,** we see that the **RBP** network with **460** iterations during step I and **100** iterations during step **II** can reach the same absolute error sum as the usual BP network with **600** training iterations. This means we need only **81%** training time with the RBP network with the LMAV rule to achieve the same performance by usual BP with 600 training iterations. In Table **5.10,** after **412** iterations during step I and 100 iterations during step **II,** the RBP network with the LMAV rule can reach the same performance **as** with the usual BP network with **600** training iterations. Therefore, the training time by the RBP network with delta rule is 76% of the training time by the usual BP network. In Table **5.8,** after **220** iterations and **100** iterations during step I and step II, **respectively,** the **RBP** with LMAV rule achieved the same performance as with the usual BP network with 600 training iterations. In this **case,** the training time of the RBP network is **42%** of the usual BP network. Fig. **5.3** shows the normalized Feigenbaum chaotic time series data versus the predicted time series data of the one-stage network (4 hidden node) with the RBP stage and the delta rule. **2000** iterations and **200** iterations were used during step I and step II, respectively. Fig. **5.4** shows the normalieed Feigenbaum chaotic time aeries data versus the predicted time **series** data of the one-atage network **(4** hidden node) with the

RBP stage and the LMAV rule. In this experiment, there were 600 training iterations during step I, and 100 iterations during step 11.

The second time series we used to test the RBP network was the Mackey-Glass time series. The Mackey-Glass equation in the discrete-time domain can be written as [Farm82]

$$x(t+1)=\frac{ax(t-\Delta)}{1+x^c(t-\Delta)}+(1-b)x(t)$$

The constant were taken to be $a=0.2$, $b=0.1$ and $c=10$. Choosing $\Delta=17$, we generated 500 data points which were used in the following experiments.

Table 5.12 shows the performance using the RBP networks with the delta rule, listed as a function of training iterations during step I. The length of input vector is 4 and 10 hidden units were used. The gain factor was 0.1 (duringstep I and 0.01 during step II. In this table, we see that the RBP network with 100 iterations during step I and 200 iterations during step II can reach a deeper minimum than the usual BP network with 1000 iterations. Therefore, we need only 14% training time with the RBP network to achieve better performance than that by the usual BP network with 1000 iterations. Table 5.13 shows the performance using the RBP network with the LMAV rule, listed as a function of training iterations during step I. The length of the input vector was 4 and 10 hidden units were used. The gain factor was 0.1 during step I and. 1.E-6 during step II. In this table, we see that the RBP network with 100 iterations during step.I and 100 iterations during,step II can reach a deeper minimum than the usual BP network with 1000 iterations. We also need only 12% training time with the RBP network with LMAV rule to achieve better performance than that by the usual BP network with 1000 iterations. Fig. 5.5 shows the original Mrrckey-Glass chaotic time series data versus the predicted time series data of the one-stage network with the RBP stage and the delta rule. 1000 iterations and 200 iterations were used during step I and step II, respectively. Fig. 5.6 shows the original Mackey-Glass chaotic time series data versus the predicted time series data of the one-stage network with the RBP stage and the LMAV rule. In this experiment, there were 1000 training iterations during step I, and 100 iterations during step II.

Ta'ble 5.1. Nonlinear Speech Prediction Performance of a One-Stage RBP Network and the LMAV Rule ($err=||e||_1$).

# of iterations		err	
step I	step II	step I	step I1
400	200	8.1649	7.6647
450	200	7.8115	7.2349
500	200	7.4919	6.9461
550	200	7.3080	6.9169
600	200	7.2612	6.8658
650	200	7.1472	6.7187

Table 5.2. Error Reduction with a Single Stage Network with 12 Hidden Units Trained by BP ($err1 = ||e_1||^2$).

# of iterations	err
1000	1.1454
2000	0.8413
5000	0.6822
10000	0.4464
20000	0.2424
30000	0.2506
40000	0.2205
50000	0.1962

Table 5.3. Error Reduction with a Two-Stage PSHNN with 6 Hidden Units per SNN Trained by Forward-Backward BP ($err1=||e_1||^2, err2=||e_2||^2$).

# of sweeps	errl	err2
20	1.0528	1.0473
40	0.8962	0.8945
100	0.6031	0.6023
200	0.4374	0.4368
300	0.3367	0.3364
400	0.2714	0.2711
500	0.2133	0.2133
600	0.1927	0.1925
700	0.1895	0.1962
800	0.1771	0.1816
900	0.1731	0.1859
1000	0.1658	0.1708

Table 5.4. Error Reduction with a Three-Stage PSHNN with 4 Hidden Units per SNN Trained by Forward-Backward BP.

# of sweep	err1f	err2f	err3f	err2b
10	1,2380	1.2157	1.2138	1.1982
50	0.6486	0.6464	0.6462	0.6447
100	0.5240	0.5236	0,5236	0.5235
200	0.4488	0.4487	0.4483	0.4484
300	0.2825	0.2823	0.2819	0.2817
423	0.1965	0.1965	0.1962	0.1962
500	0.1705	0.1704	0.1704	0.1703
600	0.1604	0.1604	0.1604	0.1603
700	0.1551	0.1551	0.1551	0.1551
750	0.1529	0.1529	0.1529	0.1529

Table 5.5. Error Reduction with a Four-Stage PSHNN with 3 Hidden Units per SNN Trained by Forward-Backward BP.

# of sweep	err1f	err2f	err3f	err4f	err3b	err2b
10	1.3594	1.3561	1.3238	1.3195	1.2963	1.2914
50	0.6716	0.6707	0.6682	0.6682	0.6662	0.6662
100	0.5121	0.5119	0.5116	0.5116	0.5115	0.5114
200	0.4136	0.4136	0.4134	0.4134	0.4134	0.4132
300	0.3540	0.3540	0.3539	0.3538	0.3538	0.3537
400	0.3093	0.3093	0.3092	0.3091	0.3090	0.3090
500	0.2620	0.2619	0.2618	0.2618	0.2618	0.2617
600	0.2306	0.2306	0.2305	0.2304	0.2303	0.2304
666	0.2210	0.2209	0.2209	0.2208	0.2208	0.2208

Table 5.6. Prediction with Feigenbaum Chaotic Time Series Data Using a 2 Hidden Node Network with the RBP Stage and the Delta Rule ($err=||e||^2$).

# of iterations		err	
step I	step II	step I	step II
100	200	1.46E-3	6.183-4
200	200	8.283-4	5.68E-4
360	200	7.153-4	5.07E-4
500	200	6.52E-4	4.78E-4
1000	200	5.57E-4	4.46E-4
1500	200	5.27E-4	4.40E-4
2000	200	5.10E-4	4.343-4

Table 5.7. Prediction with Feigenbaum Chaotic Time Series Data Using a 4 Hidden Node Network with the RBP Stage and the Delta Rule (err=$||e||^2$).

# of iterations		err	
step I	step II	step I	step II
100	200	1.71E-2	7.91E-4
120	200	1.23E-3	1.40E-4
200	200	1.91E-4	1.24E-4
500	200	1.81E-4	1.21E-4
1000	200	1.71E-4	1.19E-4
1500	200	1.65E-4	1.18E-4
2000	200	1.60E-4	1.18E-4

Table 5.8. Prediction with Feigenbaum Chaotic Time Series Data Using a 8 Hidden Node Network with the RBP Stage and Delta Rule (err=$||e||^2$).

# of iterations		err	
step I	step II	step I	step II
60	200	4.333-4	4.81E-5
100	200	8.81E-5	4.383-5
200	200	8.34E-5	4.21E-5
500	200	7.31E-5	3.87E-5
1000	200	6.34E-5	3.61E-5
1500	200	5.82E-5	3.50E-5
2000	200	5.51E-5	3.44E-5

Table 5.9. Prediction with Feigenbaum Chaotic Time Series Data Using a 2 Hidden Node Network with the RBP Stage and the LMAV Rule ($err = \|e\|_1$).

# of iterations		err	
step I	step II	step I	step II
200	100	0.5290	0.4949
300	100	0.4532	0.4206
400	100	0.4075	0.3789
452	100	0.3907	0.3619
500	100	0.3782	0.3519
600	100	0.3622	0.3336

Table 5.10. **Prediction with Feigenbaum Chaotic Time Series Data Using a 4 Hidden Node Network with the RBP Stage and the LMAV Rule** ($err = ||e||_1$).

# of iterations		err	
step I	step II	step I	step II
200	100	0.2772	0.2148
300	100	0.2340	0.1778
400	100	0.1983	0.1492
412	100	0.1945	0.1460
500	100	0.1693	0.1258
600	100	0.1462	0.1076

Table 5.11. **Prediction with Feigenbaum Chaotic Time Series Data Using a 8 Hidden Node Network with tbe RBP Stage and the LMAV Rule** $(err=||e||_1)$.

# of iterations		err	
step I	step II	step I	step II
200	100	0.2077	0.1493
220	100	0.2036	0.1410
300	100	0.1881	0.1298
400	100	0.1705	0.1174
500	100	0.1550	0.1062
600	100	0.1412	0.0965

Table 5.12. **Prediction with Mackey-Glass Chaotic Time Series Data Using a 10 Hidden Node Network with the RBP Stage and the Delta Rule** ($err=||e||^2$).

# of iterations		err	
step I	step II	step I	step II
100	200	0.7201	0.1702
200	200	0.6766	0.1621
300	200	0.6378	0.1542
500	200	0.5717	0.1393
700	200	0.5173	0.1256
900	200	0.4715	0.1130
1000	200	0.4512	0.1071

Table 5.13. **Prediction with Mackey-Glass Chaotic Time Series Data Using a 10 Hidden Node Network with the RBP Stage and the LMAV Rule** ($err = ||e||_1$).

# of iterations		err	
step I	step II	step I	step II
100	100	15.5708	9.9152
200	100	15.0856	9.6504
400	100	14.2424	9.1584
600	100	13.5287	8.7164
800	100	12.9120	8.3077
1000	100	12.3674	7.9273

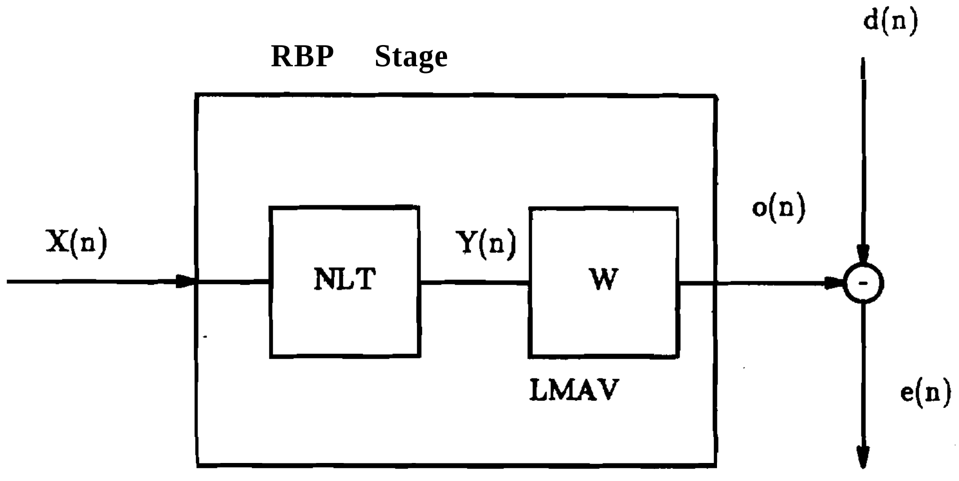

Figure 5.1. One-Stage Network with the RBP and the LMAV Rule.

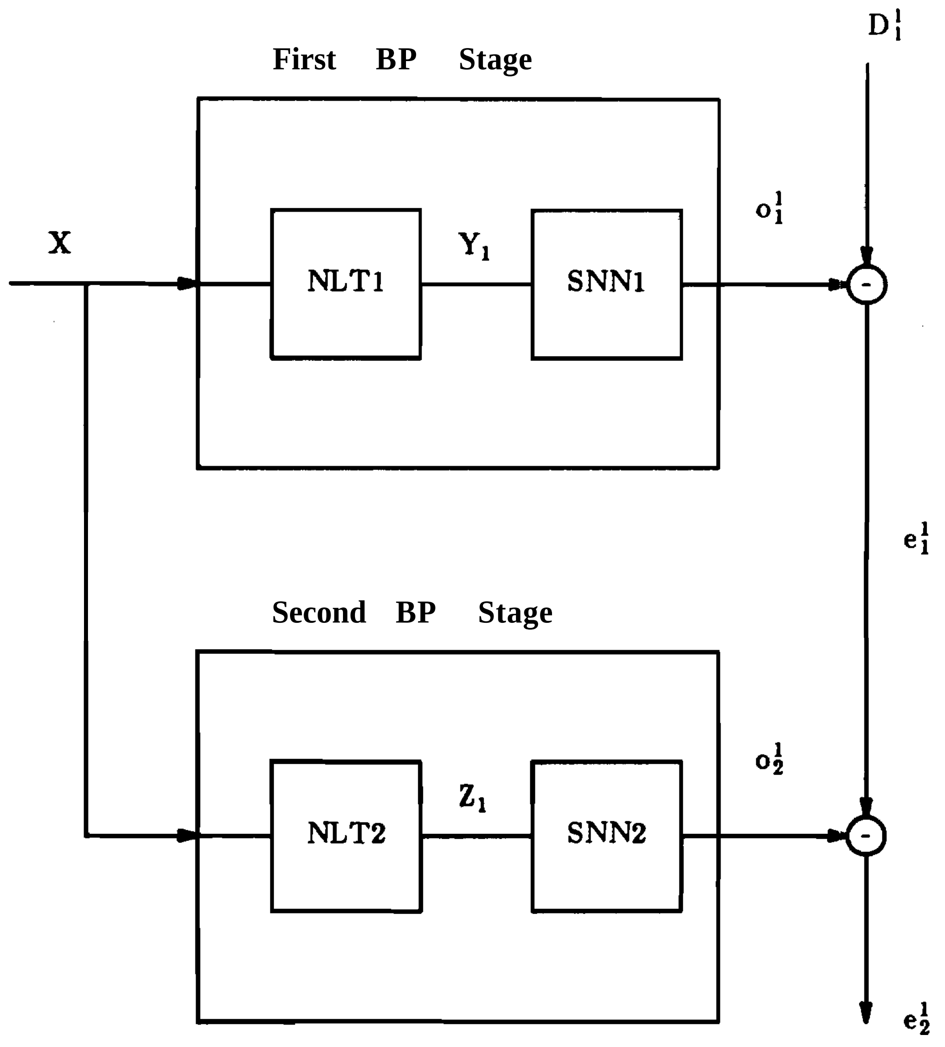

Figure 5.2. Two-Stage PSHNN with BP Stages and Forward-Backward Training.

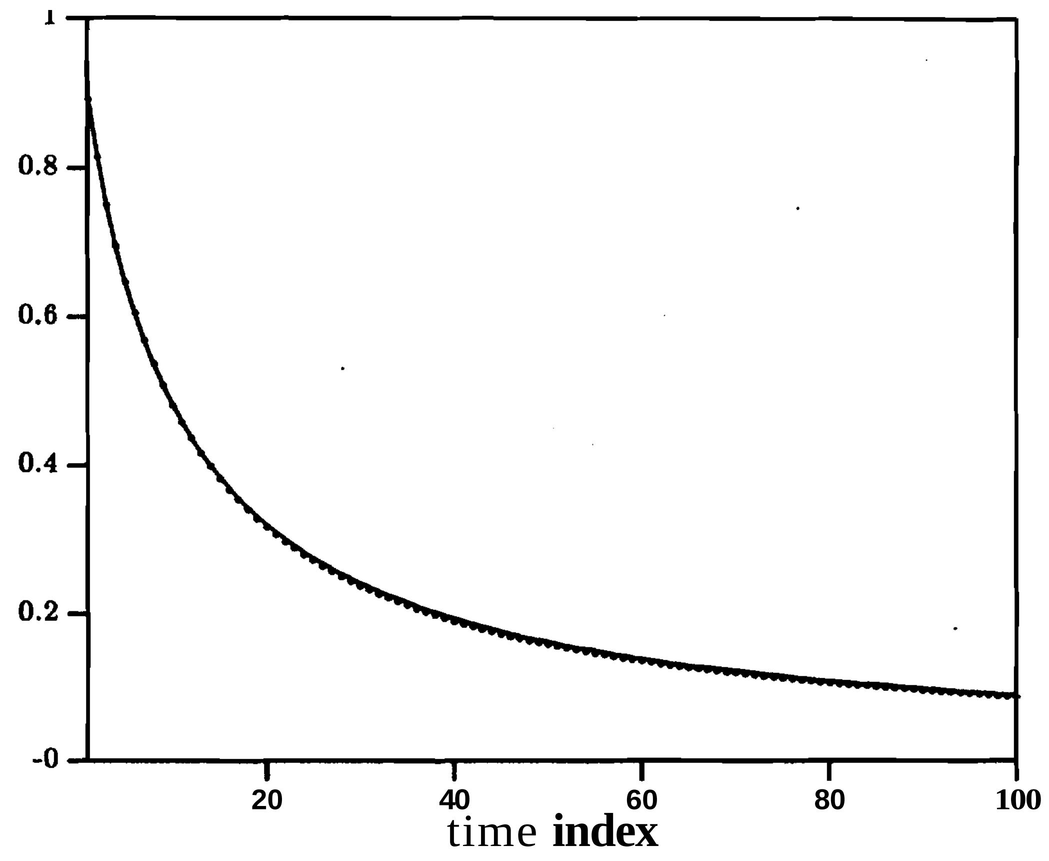

Figure 5.3. Normalized Feigenbaum Time Series (Solid Line) and the Predicted Time Series (Dotted Line) with the RBP and the Delta Rule.

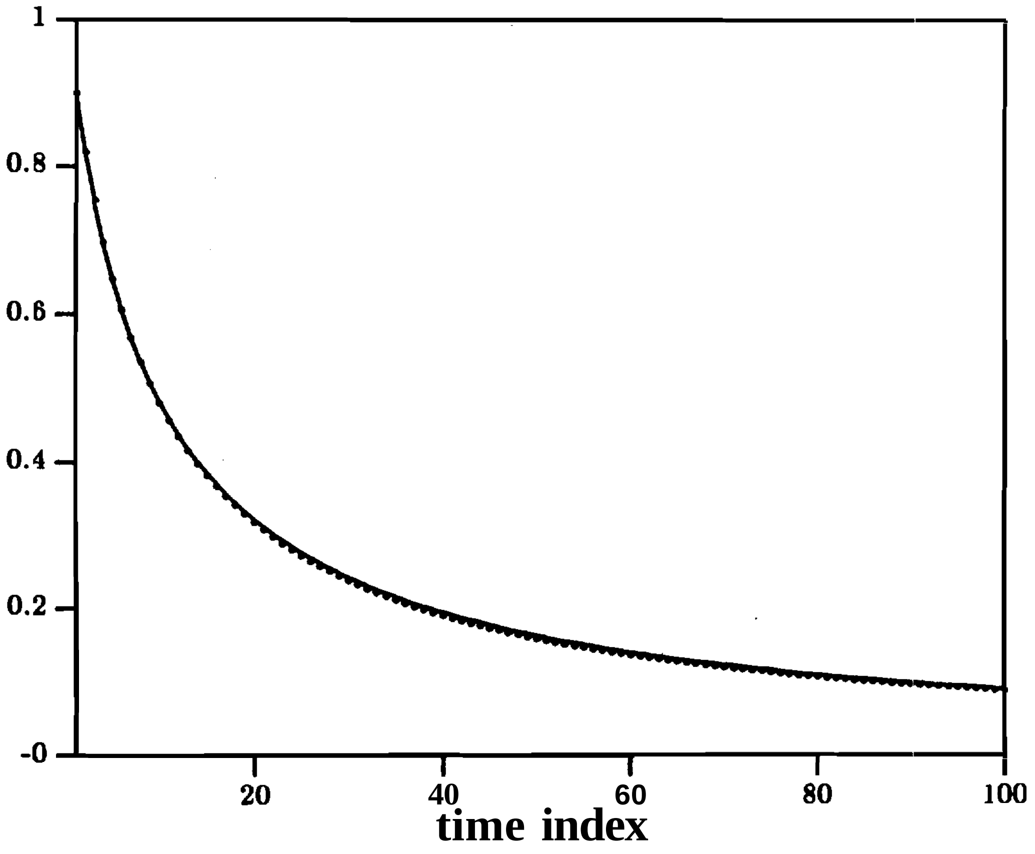

Figure 5.4. Normalized Feigenbaum Time Series (Solid Line) and the Predicted Time Series (Dotted Line) with the RBP and the LMAV Rule.

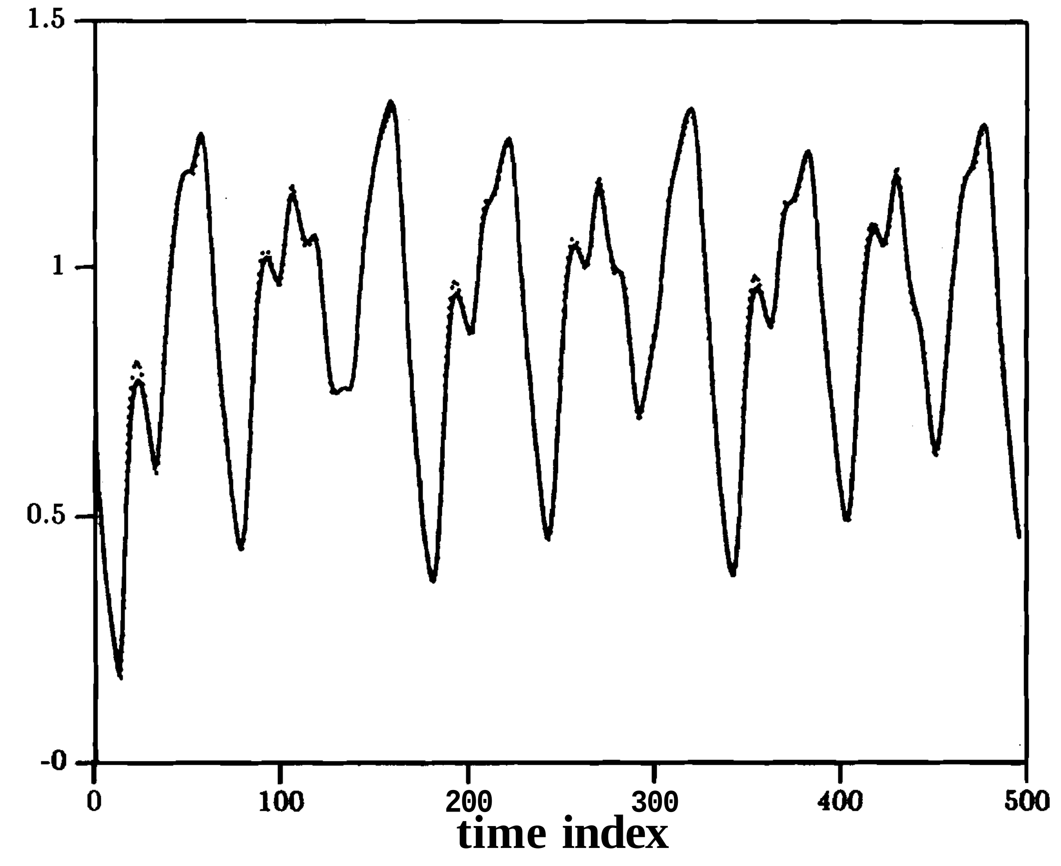

Figure 5.5. Mackey-Glass Time Series (Solid Line) and the Predicted Time Series (Dotted Line) with the RBP and the Delta Rule.

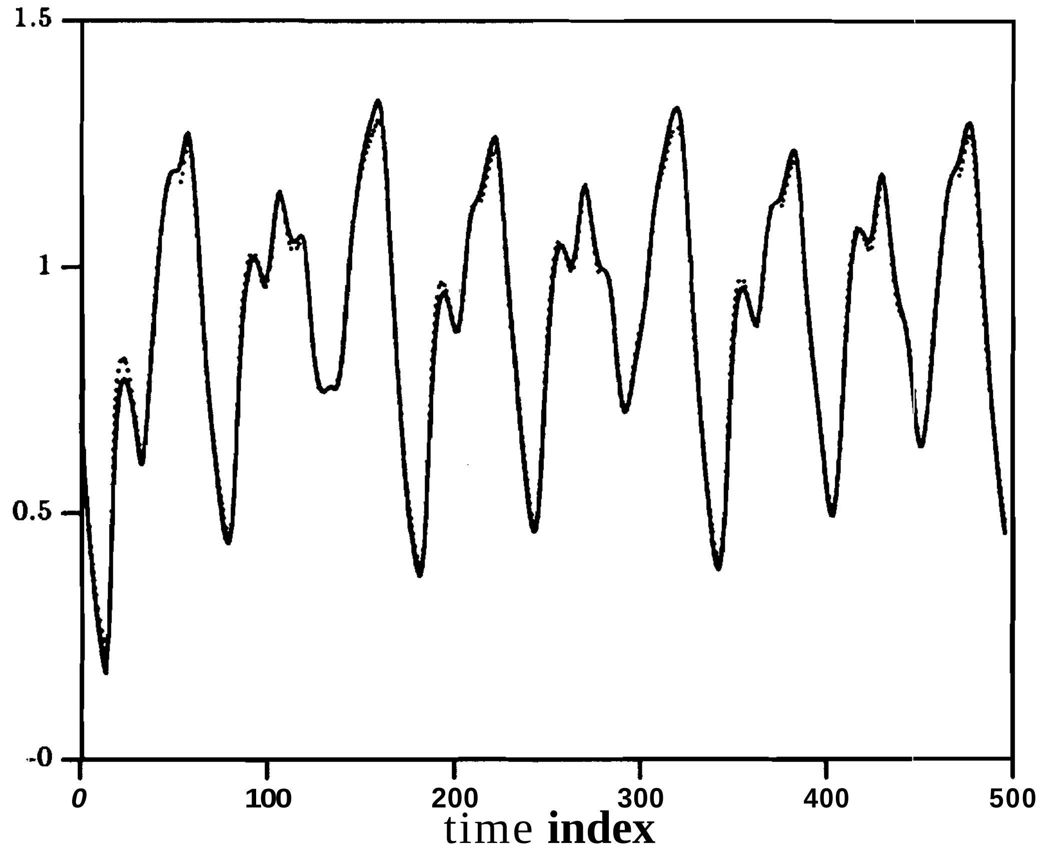

Figure 5.6. Mackey-Glass Time Series (Solid Line) and the Predicted Time Series (Dotted Line) with the RBP and the LMAV Rule.

CHAPTER 6
CONCLUSIONS

6.1. Conclusions

PSHNN's with continuous inputs and outputs have many advantages such. as error reduction, better prediction than linear prediction, parallel operation of stages, self-organizing number of stage, realizability of sequential learning, and error criterion other than mean-square error.

Computer experiments showed that linear outputs give better results when the outputs are continuous. Consequently, nonlinearities were used at other layers. In addition, linear outputs allow the use of sequential leastsquares. Even though any kind of input nonlinearity guarantees better performance over a one-stage network, the optimization of the input nonlinearities is an important issue to minimize output errors. The RBP algorithm is one effective solution to this problem. Another advantage of the RBP algorithm is that we have flexibility of choosing a different training rule due to different error criterion during step II. For example, the delta rule, the SLS and the LMAV rule can be used during step II of the RBP algorithm. Other criteria such as total least squares can also be applied.

We showed theoretically that PSHNN's with forward-backward training of n-stage networks will achieve the same error reduction as the total function-link network with the leastsquares pseudoinverse solution. In practice , experimental results show that PSHNN's in many cases have faster convergence rate and better numerical error reduction than the total function-link networks. The property that PSHNN's can divide a large size network into several smaller size networks which can learn faster and more easily in training and operate in parallel in testing is believed to be significant for real-time implementation.

We proved that the PSHNN's with any input nonlinear transformation have better performance than one-stage networks [ErDe911]. By using additional neural networks, one can learn input NLT's at every parallel stage of the PSHNN. The PSHNN with BP stages and forward-backward training is one effective solution to this problem. When backpropagation is to be used, experiments indicate that better performance in terms of a deeper minimum and convergence rate is achieved when a single BP network is replaced by a PSHNN of equal complexity in which each stage is a BP network of smaller complexity than the single BP network. With these properties, PSHNN's with continuous inputs and outputs and forward-backward training are expected to be useful in various applications of neural networks, adaptive signal processing, system identification and adaptive control.

6.2. Further Research

The following is an outline of future research topics.

(1) The proof of Theorem 4.1a has been based on n-stage PSHNN's with forward-backward training. Experimentally, we have also observed that circular training gives the same results as forward-backward training. It is desirable to give a rigorous proof for the n-stage PSHNN with circular training.

(2) The theoretical and experimental investigations so far have been carried out with stages based on the delta rule, the usual BP or the RBP. An interesting question is whether these and/or similar results are valid for stages based on other learning algorithms.

(3) The input nonlinearities may be replaced by output nonlinearities. However, we have not investigated the simultaneous use of input and output nonlinearities yet. This is especially an important problem in the case of forward-backward training. In this case, it is no longer possible to compare the PSHNN stages with forward-backward training to a single total network which converges to the pseudoinverse solution.

(4) A major consideration is whether it is possible with the forward-backward training algorithm to achieve a minimum the same as or closer to the global minimum than what other architecture yield.

(5) One important advantage of the PSHNN with continuous inputs and outputs is the ability to incorporate sequential learning so that the network continues to learn with each new input data without requiring the storage of past information. This has been implemented with stages without forward-backward training. It is desirable to apply SLS learning with forward-backward training as well as more complex networks.

(6) Another important problem is how to optimize input and/or output nonlinearities. It is desirable to have simple, pointwise nonlinearities for real-time implementation, and they should be learned, probably adaptively in time,

for optimal performance. It is possible to incorporate fast transforms in addition to pointwise nonlinearities as preprocessing to the network. The fast transforms provide a number of advantages such as feature selection, achieving invariance to a number of distortions like translation, rotation and scaling, and minimizing network size.

(7) The theoretical and experimental results obtained are mostly with respect to the mean-square error criterion. We have also developed the method which uses the LMAV rule during step II of the RBP stages. Other error criteria such as weighted leastsquares and total leastsquares during step II of the RBP stages should be investigated.

(8) An interesting area in systems and signal processing is system modeling and identification. Neural networks with nonlinear activation functions are an effective way to construct a model for the transfer function of an unknown system with only a finite data set of inputs, and associated outputs of the system. Techniques concerning nonlinear system modeling by PSHNN's are expected to be useful in spectral estimation, biomedical signal modeling, and other applications. Further studies need to be carried out on such topics.

LIST OF REFERENCES

[AgEr91] S. Aghagolzadeh, O. K. Ersoy, "Optimal Multistage Transform Image Coding", IEEE *Tran. Circuits and Systems for Video Technology,* December 1991.

[Alex86] S. T. Alexander, *"Adaptive Signal Processing, Theory and Applications",* Springer-Verlag, New York, pp. 68-85, 1986.

[Bell87] M. G. Bellanger, *"Adaptive Digital Filter and Signal Analysis",* Maurice1 Dekker Inc., pp. 114-121, 1987.

[DeEr91] S-W. Deng, O. K. Ersoy, "Parallel, Self-Organizing, Hierarchical Neural Networks with Circular Training", *Purdue University Tech. Report* No. TR-EE-91-16, April 1991.

[DeEr921] S-W. Deng, O. K. Ersoy, "F'arallel, Self-Organizing, Hierarchical Neural Networks with Forward-Backward Training*,submitted to *Circuits, Systems and Signal Processing,* January 1992.

[DeEr922] S-W. Deng, O. K. Ersoy, "Parallel, Self-Organizing Neural Networks for Nonlinear Prediction, Filtering and System Identification", submitted to *IEEE Tran. Neural Networks,* 1992.

[DuHa73] R. O. Duda, P.E. Hart, *"Pattern Classification and Scene Analysis",* John Wiley & Sons Inc., pp. 159-162, 1973.

[ErDe911] O. K. Ersoy and S-W. Deng, "Parallel, Self-Organieing, Hierarchical Neural Networks with Continuous Inputs and Outputs", *Proc. Hawaii Int. Conf. System Sciences,* HICCS-24, pp. 486-492, Kauai, January 1991.

[ErDe912] O. K. Ersoy and S-W. Deng, "Parallel, Self-Organizing, Hierarchical Neural Networks, with Continuous Inputs and Outputs", *Purduc University Tech. Report,* No. TR-EE-81-51? December 1991, and to appear in *IEEE Tran. Neural Networks.*

[ErHo90] O. K. Ersoy, D. Hong, *Parallel, Self-Organizing, Hierarchical Neural Networks", *IEEE Trans. Neural Networks,* Vol. 1, No. 2, pp. 167-178, June 1990.

[ErHoII] O. K. Ersoy, D. Hong, "Parallel, Self-Organizing, Hierarchical Neural Networks II", to appear in *IEEE Tran. Industrial Electronics,* Special lssue on Neural Networks.

[Erso88] O. K. Ersoy, "A Study of Associative Memory Based on the Delta Rule", *IEEE Int. Conf. Neural Networks* , San Diego, Calif., July 1988.

[ErZB90] O. K. Ersoy, J. Y. Zhuang, J. Brede, "An Iterative Interlacing Approach to the Synthesis of Computer-Generated Holograms", *Purdue University Tech. Report,* No. TR-EE-90-59, November 1990, and submitted to *Applied Optics.*

[Farm82] J. D. Farmer, "Chaotic Attractors of an Infinite-Dimensional Dynamical System", *Physica* D, Vol. *D* 4, pp. 366-393, 1982.

[Feig78] M. Feigenbaum, "Quantitative Universality for a Class of Nonlinear Transformations", *J. Statistical Physics,* Vol. 19, pp. 25-52, 1978.

[GiMa87] C. L. Giles, T. Maxwell, "learning, Invariance and Generalization in Higher Order Networks," *Applies Optics,* Vol. 26, No.23, pp. 4972-4978, December 1987.

[Grau84] D. Graupe, *"Time Series Analysis, Identification, and Adaptive Filtering',* Robert F. Krieger, 1984.

[HaDr82] R. E. Hartwig and M. P. Drazin, 'Lattice Properties of the *-Order for Complex Matrices", *J. of Math. Analysis and Applications,* Academic Press, Inc. 1982.

[Hake75] H. Haken, "Analogy between Higher Instabilities in Fluids and Lasers", *Physics Letters,* Vol. A53, pp. 77-78, 1975.

[Hayk91] Simon Haykin, *"Adaptive Filter Theory",* 2nd ed., Prentice-Hall, Inc., pp. 299-341, 1991.

[Hong90] D. Hong, "Parallel, Self-Organizing, Hierarchical Neural Networks", Ph.D. Dissertation, Purdue University, August 1990.

[HoKu71] K. Hoffman, R. Kunze, *"Linear Algebra",* 2nd ed., Prentice-Hall, Inc., p. 211, 1971.

[JaMF84] S. C. Jacobsen, S. G. Meek, R. R. Fullmer, "An Adaptive Myoelectric Filter", *6th* IEEE *Conf. Eng. in Med. and Biol. Soc.,* 1984.

[Kell90] M. F. Kelly, "The Application of Neural Networks to Myoelectric Signal Analysis: A Preliminary Study", *IEEE Transaction on Biomedical Engineering,* **Vol. 37** No. 3, March **1990.**

[KoPo85] T. Koh, E. J. Powers, "Second-Order **Volterra** Filtering and its Application to Nonlinear System Identification", *IEEE Tran. on ASSP,* Vol. ASSP-33, No. 6, pp. **1445-1455,** December 1985.

[LaFa87] A. Lapedes, R. Farber, "Nonlinear **Processing** Using Neural Networks: Prediction and System Modeling", *Los Alamos National Laboratory,* **LA-UR-87-2662,** 1987.

[Luen84] D. G, Luenberger, *"Introduction to Linear and Nonlinear Programming",* Addison-Wesley Pub. Company, second edition, pp. 227-230, 1984.

[Math91] V. J. **Mathews, "Adaptive** Polynomial Filter", *IEEE Signal Processing Magazine,* **Vol.** 8, No. 3, pp. **10-26,** July 1991.

[Mend73] J. M. **Mendel,** *"Discrete Techniques of Parameter Estimation",* Marcel Dekker, pp. 91-107, 1973.

[Miya88] Irie, Miyake, "Capabilities of Three-Layered Perceptrons", *Proc. IEEE ICNN,* Vol. 1, pp. **641-648,** San **Diego,** July **1988.**

[MoTu77] F. Mosteller, J. Tukey, *"Data Analysis and Regression: a Second Course in Statistics",* Addison-Wesley Publishing Company, pp. **365-369,** 1977.

[Naka53] H. Nakano, *"Spectral Theory in the **Hilbert** Space",* Japan Society for the Promotion of Science, 1953.

[Pars86] T. W. Parson, *"Voice and Speech Processing",* McGraw-Hill, **pp.138-145,** 1986.

[Pao89] Y-M. **Pao,** *"Adaptive Pattern Recognition and Neural Networks",* Addison-Wesley Pub. Company, Inc., 1989.

[Pyle67] L. D. Pyle, "A Generalieed Inverse **ϵ-Algorithm** for Constructing **Intersection** Projection Matrices with Applications", ***Numerische*** *Mathematik **10,*** pp **86-102,** 1967.

[RaMi71] C. R. Rao, S. K. Mitra, *"Generalized Inverse of Matrices and its **Applications**",* John Wiley & Sons, Inc., pp. **106-107,** 1971.

[RuHO80] D. Russell, J. **Hanson,** E. Ott, **"Dimension** of Strange Attractors", *Physical Review Letters,* Vol. 45, *pp.* 1175-1178, 1980.

[Rume88] D. E. Rumelhart, *"Parallel Distributed Processing"*, The MIT Press, Cambridge Mass. , 1988.

[RuTa71] D. Ruell, F. Takens, "On the Nature of Turbulence", *Communications in Mathematical Physics,* Vol. 20, pp. 167-192, 1971.

[Sore85] H. W. Soremen, *"Parameter Estimation, Principles and Problems",* M. Dekker, New York, 1985.

[Stra86] G. Strang, *"Linear Algebra and its Applications",* Gilbert Strang, third edition, 1986.

[SwGo78] H. Swinney, J. P. Gollub, "The Transition for Turbulence", *Physics Today,* Vol. 45 PP 41-49, August 1978.

[ToKa79] K. Tomita, T. Kai, "Chaotic Response of a Limit Cycle", *J. Statistical Physics,* Vol. 21, pp. 65-86, 1979.

[WiHo60] G. Widrow, M. E. Hoff, "Adaptive Switching Circuits," *Inst. Radio Engineers Western Electronic Show and Convention Record,* Part 4, pp. 96-104, 1960.