Nonlinear Adaptive Prediction of Nonstationary Signals

Simon Haykin, *Fellow, IEEE*, and Liang Li

***Abstract*—In this paper, we describe a computationally efficient scheme for the nonlinear adaptive prediction of nonstationary signals whose generation is governed by a nonlinear dynamical mechanism. The complete predictor consists of two subsections: One performs a nonlinear mapping from the input space to an intermediate space with the aim of linearizing the input signal, and the other performs a linear mapping from the new space to the output space. The nonlinear subsection consists of a pipelined recurrent neural network (PRNN), and the linear section consists of a conventional tapped-delay-line (TDL) filter. The nonlinear adaptive predictor described herein is of general application. The dynamic behavior of the predictor is demonstrated for the case of a speech signal; for this application, it is shown that the nonlinear adaptive predictor outperforms the traditional linear adaptive scheme in a significant way.**

I. Introduction

THE prediction of a time series is synonymous with modeling of the underlying physical mechanism responsible for its generation. Many of the physical signals encountered in practice exhibit two distinct characteristics: nonlinearity, and nonstationary. Consider, for example, the important case of speech signals. It is well known that the use of prediction plays a key role in the modeling and coding of speech signals [1]. The production of a speech signal is known to be the result of a dynamic process that is both nonlinear and nonstationary. To deal with the nonstationary nature of speech signals, the customary practice is to invoke the use of adaptive filtering. However, the nonlinear nature of the speech production process has not received the attention it deserves in that much of the literature on the prediction of speech signals has focused almost exclusively on the use of linear adaptive filtering schemes [2]. Yet, one of the classic papers written by Gabor [3] on a learning machine for the adaptive prediction of speech signals over 30 years ago emphasized the use of nonlinear processing, exemplified by what we now refer to as the Volterra series.

A neural network is well suited for the nonlinear prediction of nonstationary signals by virtue of the distributed nonlinearity built into its design and the ability of the network to learn from its environment. The key question is how to design such a network for the nonlinear prediction of nonstationary signals. The traditional method of supervised learning is unsuitable because of its off-line training requirement. What we need is a neural network that is able to learn in an on-line fashion, that is, on the fly, in which case, the network learns to adapt to statistical variations of the incoming time series while performing its filtering role at the same time. This form of learning, which is referred to as *in-situ* learning or continuous learning, can indeed be a difficult task. In this paper, we describe a computationally efficient neural network for the nonlinear adaptive prediction of nonstationary signals [4] and demonstrate its application to a speech signal. It should, however, be emphasized that the network has a much wider range of applications such as system identification, adaptive equalization, and adaptive noise cancellation. We have chosen the case of speech signals in this paper merely as a case study. An early version of the new neural network was first described in [5].

The paper is organized in the following manner. In Section II, we present motivation for the new neural network. The structure of the network is both modular and recurrent, the formulation of which is motivated by the principle of divide and conquer. In Section III, we present a detailed description of the network used, in conjunction with a conventional tapped-delay line filter, as a one-step predictor; the algorithmic design of the network is summarized in an appendix at the end of the paper. In Section IV, we present an experimental study of the new network applied to the prediction of a speech signal. This is followed by a comparison of the network's performance with that of two single sections: a) linear predictor of finite-duration impulse response and b) conventional recurrent neural network. The paper concludes with some final remarks in Section V.

II. Motivation

In the introductory section, we emphasized the need for continuous learning when the task at hand involves the adaptive nonlinear prediction of a nonstationary time series (e.g., speech signal). A learning algorithm that is well suited for such a task is the so-called real-time recurrent learning (RTRL) algorithm described by Williams and Zipser [6], aspects of which may be traced back to McBride and Narendra [7]. This algorithm is used to train a fully connected recurrent network in which the output signals of the output neurons and the hidden neurons are all fed back to the input layer. Important advantages of the RTRL algorithm include the following:

- The algorithm is capable of nonlinear adaptive filtering of nonstationary signals.

Manuscript received November 28, 1993; revised June 20, 1994. This work was suported by the Natural Sciences and Engineering Research Council of Canada. The associate editor coordinating the review of this paper and approving it for publication was Prof. J. N. Hwang.

The authors are with the Communications Research Laboratory, McMaster University, Hamilton, Canada L8S 4K1.

IEEE Log Number 9407620.

- The algorithm does not require *a priori* knowledge of time dependences among the input data.

However, a major limitation of the RTRL algorithm is that its computational complexity increases as $O(N^4)$, where N is the total number of neurons in the network. When the learning task of interest is a difficult one, N may assume a large value making the computational requirements of the algorithm unacceptable. Indeed, this is the sole reason that has hindered large-scale applications of the RTRL algorithm in its conventional form.

In this paper, we describe a *pipelined recurrent neural network* (PRNN), the algorithmic design of which builds on the RTRL algorithm. The new network is so called because of its modular and recurrent structure. It is the modularity of the network that helps us contain the computational complexity of the RTRL algorithm, as will be explained later.

The design of the new network structure follows an important engineering principle, namely, the *principle of divide and conquer*, which may be stated as follows:

- To solve a complex problem, break it into a number of simpler problems.

According to Van Essen *et al.* [8], this same principle is also reflected in the design of the brain, as summarized here in the context of the primate visual system:

- Separate modules are created for different subtasks, permitting the neural architecture to be optimized for particular types of computation.
- The same module is replicated several times over.
- A coordinated and efficient flow of information is maintained between the modules.

Modularity, as an important principle of learning, is also emphasized by Houk [9] and other investigators. Now, in a loose sense, all three elements of the principle of divide and conquer, viewed in a biological context, feature in the PRNN, as explained here:

- The PRNN is composed of M modules, each of which is designed to perform nonlinear adaptive filtering on an appropriately delayed version of the input signal vector.
- The M modules of the PRNN are identical, and each is designed as a fully connected recurrent network with a single output neuron.
- Information flow into and out of the modules proceeds in a synchronized fashion.

III. Nonlinear Adaptive Predictor

A. Principle of Operation

Fig. 1 shows a block diagram of the complete nonlinear adaptive filtering system. The PRNN, consisting of many levels of recurrent signal processing, constitutes the nonlinear subsection of the system. In addition, the system includes a linear subsection represented by a conventional tapped-delay-line (TDL) filter. These two subsections perform distinct functions of their own, as illustrated in Fig. 2:

- The PRNN performs a nonlinear mapping from the input space to an intermediate space with the aim of linearizing the input signal. It does so by filtering a set of samples of the input signal $s(t)$, which are represented by $s(n)$, $s(n-1)$, $s(n-2), \cdots$. These samples extend into the "infinite past" for varying discrete-time n by virtue of the feedback built into the design of each module of the PRNN. Thus, the PRNN has *infinite memory* of a fading nature.
- The TDL filter performs a linear mapping from the new intermediate space to the output space. Specifically, it uses a linear combination of the samples $y(n)$ $y(n-1), \cdots, y(n-q+1)$ derived from the output of the PRNN to produce a prediction $\hat{s}(n+1)$ of the original signal one step into the future. In contrast with the PRNN, the TDL filter has *finite memory*.

Both of these operations are performed adaptively on a continuous basis. Thus, the cascade combination of the PRNN and the TDL filter may be used to perform nonlinear adaptive prediction of a nonstationary signal.

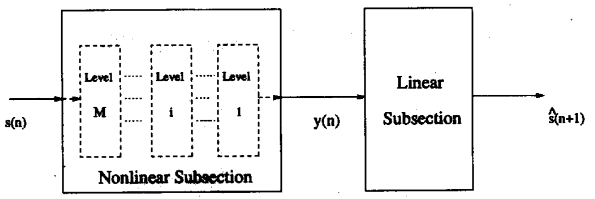

Fig. 1. Block diagram of the new nonlinear adaptive filter.

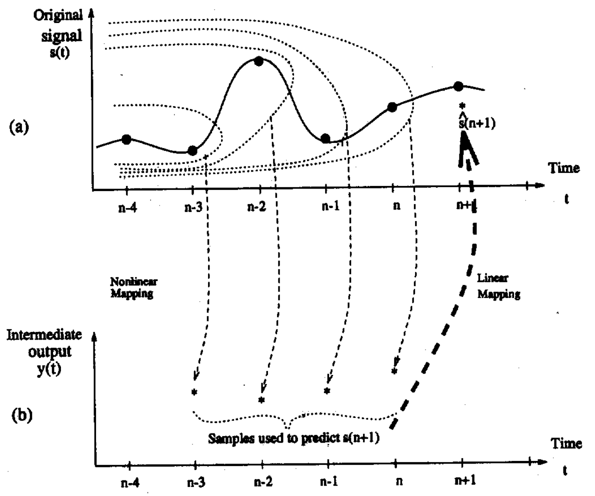

Fig. 2. Illustrating the interrelationships between the original signal $s(t)$ and the intermediate output $y(t)$.

B. Nonlinear Subsection

A detailed structure of the PRNN is shown in Fig. 3, involving a total of M levels of processing. Each level has a neural module and a comparator of its own. Every module is a dynamical submodel of the outside world. Specifically, the module consists of a fully connected recurrent neural network with N neurons. Fig. 4 shows the detailed structure of module i. In addition to the p external inputs, there are N feedback

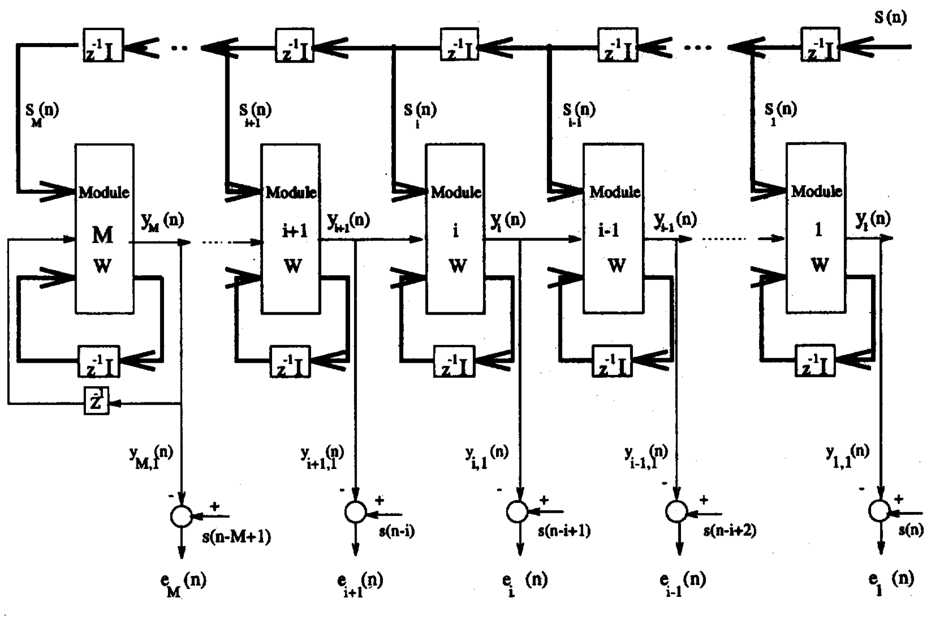

Fig. 3. Nonlinear subsection: A pipelined recurrent neural network.

signals. To accommodate a bias for each neuron, besides the $p + N$ inputs, we have included one other input whose value is always maintained at +1. Each module has $N-1$ outputs fed back to its input, and the remaining output is applied directly to the next module. In the case of module M, a one-unit delayed version of the module's output is also fed back to the input. Thus, all the modules operate similarly in that they all have exactly the same number of external inputs and feedback signals, which are properly timed. Moreover, all the modules of the PRNN are designed to have exactly the same synaptic weight matrix.

The activation function of every neuron in each module is a sigmoidal function described by the logistic function [10]

$$y_{i,k}(n) = \phi(v_{i,k}(n)) = \frac{1}{1+\exp(-v_{i,k}(n))}, \quad i = 1,\cdots,M; \quad k = 1,\cdots N \tag{1}$$

where $v_{i,k}(n)$ is the net internal activation of the kth neuron, and $y_{i,k}(n)$ is the output of the kth neuron, both referring to the ith module at the nth time point.

Let $\boldsymbol{W}$ denote the $(p+N+1)$-by-N synaptic weight matrix for each module. An element $w_{k,l}$ of this matrix represents the weight of the connection to the kth neuron from the lth input node. The weight matrix $\boldsymbol{W}$ may thus be written as

$$\boldsymbol{W} = [\boldsymbol{w}_1,\cdots,\boldsymbol{w}_k,\cdots,\boldsymbol{w}_N] \tag{2}$$

where $\boldsymbol{w}_k$ is a $(p+N+1)$-by-1 vector defined by

$$\boldsymbol{w}_k = [w_{k,1}, w_{k,2},\cdots,w_{k,p+N+1}]^T \tag{3}$$

The superscript T denotes transposition.

Suppose now that we are given a time series consisting of the observation samples $s(1), s(2),\cdots,s(n)$, referring to an input signal $s(t)$. At the nth time point, the external input applied to module i is described by the p-by-1 vector

$$\boldsymbol{s}_i(n) = [s(n-i), s(n-(i+1)),\cdots,s(n-(i+p-1))]^T \tag{4}$$

where p is the nonlinear prediction order. The other input vector applied to module i is the N-by-1 feedback vector

$$\boldsymbol{r}_i(n) = [r_{i,1}(n), r_{i,2}(n),\cdots,r_{i,N}(n)]^T. \tag{5}$$

As mentioned previously, each neuron also has a fixed input of +1, applying an adjustable bias to its activation function.

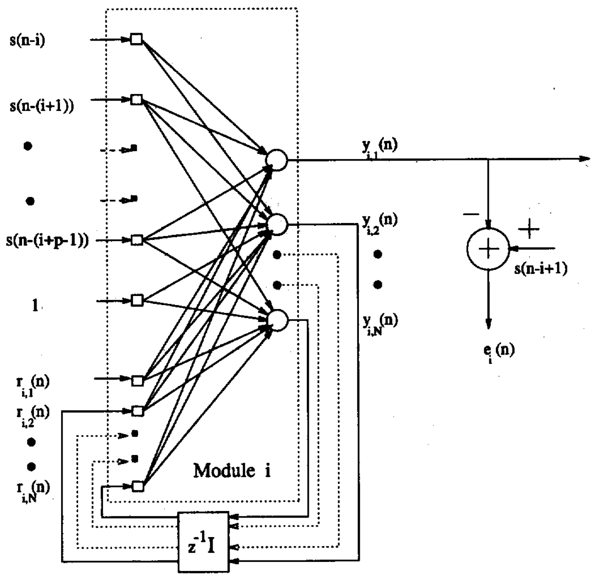

Fig. 4. Detailed construction of level i of the PRNN.

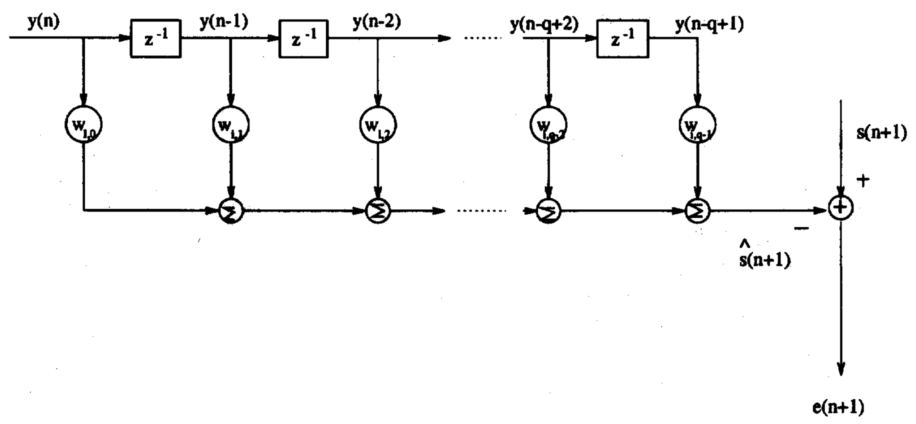

Fig. 5. Linear subsection: Tapped-delay-line filter.

At the nth time point, the output $y_{i,k}(n)$ of neuron k in module i is described by

$$y_{i,k}(n) = \phi(v_{i,k}(n)) \tag{6}$$

$$v_{ik}(n) = \sum_{l=1}^{p} w_{k,l}s(n-(i+l-1)) + w_{k,p+1} + \sum_{l=p+2}^{p+N+1} w_{k,l}r_{i,l-(p+1)}(n) \tag{7}$$

where the weight $w_{k,p+1}$ represents the adjustable bias; the index i = 1, 2, $\cdots$, M, and k = 1, 2, $\cdots$, N.

The "feedback" signals of module i consist of the first neuron's output in the adjacent module i + 1 and the one-step delayed output signals of neurons 2, $\cdots$, N in module i that are fed back to itself. We may thus define the N-by-1 feedback vector $\boldsymbol{r}_i(n)$ of the ith module as

$$\begin{aligned}\boldsymbol{r}_i(n) &= [y_{i+1,1}(n), \boldsymbol{r}_i'(n)]^T\\ &= [y_{i+1,1}(n), y_{i,2}(n-1), \cdots, y_{i,N}(n-1)]^T,\\ &\qquad i = 1, \cdots, M-1\end{aligned} \tag{8}$$

where $\boldsymbol{r}_i'(n)$ denotes those feedback signals that originate from module i itself. The last module of the PRNN, namely, module M, operates as a standard fully connected recurrent neural network [10]. The vector $\boldsymbol{y}_M(n)$, consisting of the outputs of the output neuron and all hidden neurons in this module, is fed back to itself as the feedback signal vector after a delay of one time unit, as shown by

$$\boldsymbol{r}_M(n) = \boldsymbol{y}_M(n-1). \tag{9}$$

The prediction computed by the PRNN at time n is defined by the output of the visible (first) neuron of module 1 as shown by

$$y_{\text{pred}}(n) = y_{1,1}(n) \tag{10}$$

The nonlinearity responsible for this computation is of a "nested" kind that characterizes the way in which the M modules of the PRNN are chained together. Table I is a summary of the modules' inputs and outputs involved in the computation.

TABLE I
SUMMARY OF THE INPUTS AND OUTPUTS OF THE MODULES IN THE PIPELINED RECURRENT NEURAL NETWORK DURING THE PREDICTION STAGE

	Module 1	Module i $1<i<M$	Module M
External inputs	$s(n-1)$	$s(n-i)$	$s(n-M)$
	$\vdots$	$\vdots$	$\vdots$
	$s(n-p)$	$s(n-(i+p-1))$	$s(n-(M+p-1))$
	1	1	1
Feedback signals	$y_{2,1}(n)$	$y_{i+1,1}(n)$	$y_{M,1}(n-1)$
	$y_{1,2}(n-1)$	$y_{i,2}(n-1)$	$y_{M,2}(n-1)$
	$\vdots$	$\vdots$	$\vdots$
	$y_{1,N}(n-1)$	$y_{i,N}(n-1)$	$y_{M,N}(n-1)$
Outputs	$y_{1,1}(n)$	$y_{i,1}(n)$	$y_{M,1}(n)$

The actual output of the PRNN is the filtered version of $y_{1,1}(n)$, which is denoted by $y_{\text{filt}}(n)$. This latter signal is designed to extract the full information content of the original signal $s(t)$, up to and including time n; more will be said on the computation of $y_{\text{filt}}(n)$ in Section III-E.

C. Linear Subsection

The linear subsection of the neural network-based predictor consists of a tapped-delay-line (TDL) filter, which is shown in Fig. 5. The weight vector of this filter is denoted by

$$\boldsymbol{w}_l = [w_{l,0}, w_{l,1}, \cdots, w_{l,q-1}]^T \tag{11}$$

where q is the total number of taps. The tap inputs of the linear subsection consist of the present output $y_{\text{filt}}(n)$ computed by the PRNN and $q-1$ past values of it, as shown by

$$\boldsymbol{y}_{\text{filt}}(n) = [y_{\text{filt}}(n), y_{\text{filt}}(n-1), \cdots, y_{\text{filt}}(n-q+1)]^T \tag{12}$$

where $y_{\text{filt}}(n)$ is the filtered version of $y_{1,1}(n)$. The output of the linear filter is thus defined as the inner product

$$\hat{s}(n+1) = \boldsymbol{w}_l^T \boldsymbol{y}_{\text{filt}}(n). \tag{13}$$

The output $\hat{s}(n+1)$ is a prediction of the actual sample $s(n+1)$ of the original signal $s(t)$.

D. Cost Function

Each level of the PRNN computes an error signal defined by

$$e_i(n) = s(n-i+1) - y_{i,1}(n), \qquad i = 1, 2, \cdots, M \tag{14}$$

where the sample $s(n-i+1)$ of the input signal $s(t)$ is the desired response of module i. Note that the output $y_{i,l}(n)$ of module i is limited in amplitude to the range (0, 1) by virtue of the sigmoidal activation function described in (1). The corresponding speech sample $s(n-i+1)$ is adjusted to occupy the same amplitude range prior to processing. Given the error signal $e_i(n)$, the overall cost function for the PRNN is thus defined by

$$\mathcal{E}(n) = \sum_{i=1}^{M} \lambda^{i-1} e_i^2(n) \tag{15}$$

where λ is an exponential weighting factor that lies in the range $(0 < \lambda \leq 1)$. The term λ^{i-1} is, roughly speaking, a measure of the memory of the individual modules in the PRNN. The use of this exponential weighting function is motivated by recursive lease-squares estimation [11].

E. Algorithmic Design of the Predictor

The cost function $\mathcal{E}(n)$ is minimized by using an approximation to the method of steepest descent. Specifically, the change $\Delta \boldsymbol{W}$ applied to the synaptic weight matrix $\boldsymbol{W}$ of a module is computed along the negative of the gradient of $\mathcal{E}(n)$ with respect to $\boldsymbol{W}$. Thus, the computations performed by the PRNN at time n proceed in a self-organized manner in three stages as follows:

1) *Prediction.* Given the input vectors $s_1(n)$, $s_2(n), \cdots, s_M(n)$ and the desired responses $s(n)$, $s(n-1), \cdots, s(n-M+1)$, the individual modules of the PRNN compute the one-step prediction errors $e_1(n)$, $e_2(n), \cdots, e_M(n)$ respectively, as described in (14); the vector $\boldsymbol{s}_i(n)$ is defined in (4) for $i = 1, 2, \cdots, M$.
2) *Weight Updating.* The prediction errors are used to compute the matrix of local gradients $\partial\mathcal{E}(n)/\partial\boldsymbol{W}$ and, therefore, the correction $\Delta\boldsymbol{W}$ applied to the old weight matrix $\boldsymbol{W}$. Accordingly, the updated value of the synaptic weight matrix $\boldsymbol{W}_{\text{new}}$ is computed.
3) *Filtering.* The updated weight matrix $\boldsymbol{W}_{\text{new}}$ and the updated p-by-1 input vectors with leading elements $s(n)$ $s(n-1), \cdots, s(n-M+1)$ are used to compute the filtered output $y_{\text{filt}}(n)$ as the output of the first (visible) neuron in module 1 of the PRNN. This is done by proceeding through the M modules one by one.

As mentioned previously, the filtered signal $y_{\text{filt}}(n)$ constitutes the final output of the PRNN at time n.

Turning next to the linear subsection, the output $\hat{s}(n+1)$ of the TDL filter, which is produced in response to the sequence $\{y_{\text{filt}}(n), y_{\text{filt}}(n-1), \cdots, y_{\text{filt}}(n-q+1)\}$, is compared against the desired response $s(n+1)$. The well-known least-mean-square (LMS) algorithm is used to adjust the tap weights of the TDL filter. This is done in an attempt to minimize the error signal (i.e., the difference between $s(n+1)$ and $\hat{s}(n+1)$ in a "mean-square sense").

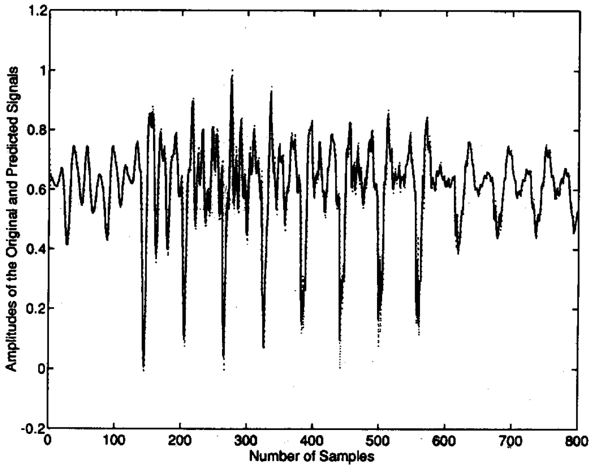

Fig. 6. Nonlinear prediction of a speech signal. Continuous curve: actual speech signal. Dashed curve: nonlinear prediction (including tapped-delay-line filtering).

A summary of the complete learning algorithm used to design the nonlinear adaptive predictor is presented in the appendix at the end of the paper. The appendix also includes a procedure for the initialization of the learning algorithm.

IV. Experimental Results

A. Nonlinear Prediction of a Speech Signal

In this section, we illustrate the application of the nonlinear adaptive prediction described herein to the important case of speech signals. In particular, we present highlights of an experimental study based on a male speech, i.e., when we record audio data .... The recorded time series corresponding to this speech signal (sampled at 8 kHz) is made up of 10 000 points.

The predictor has the following parameters:

a) Nonlinear subsection:

Number of modules, $M = 5$.

Number of neurons per module, $N = 2$.

Nonlinear prediction order, $p = 4$.

Learning rate, $\eta = 0.0001$

Forgetting factor, $\lambda = 0.9$.

b) Linear subsection:

Length of tapped-delay-line filter, $q = 12$.

Learning rate, $\mu = 0.3$.

Fig. 6 shows a plot of 800 samples of the speech signal versus time. The continuous curve is the actual speech signal, and the dashed curve is the one-step prediction performed by the nonlinear adaptive predictor consisting of the PRNN and TDL sections.

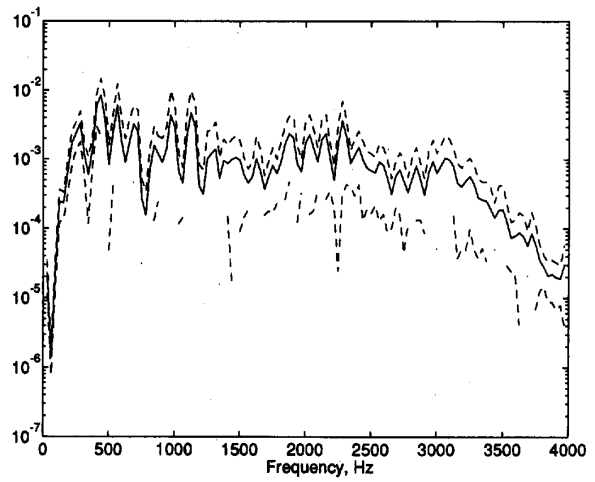

Fig. 7. Power spectrum of the nonlinear prediction error. Continuous curve: typical result. Dashed curves: upper and lower bounds representing 95% confidence.

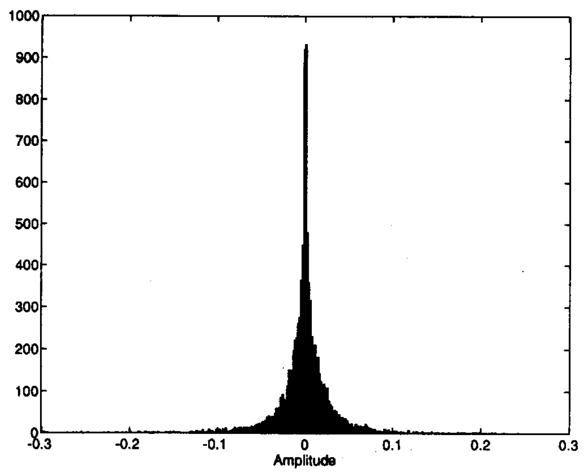

Fig. 8. Histogram of the nonlinear prediction error.

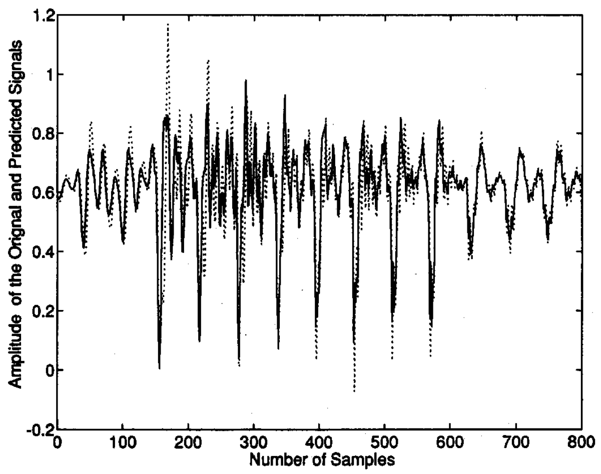

Fig. 9. Linear prediction of a speech signal. Continuous curve: actual speech signal. Dashed curve: linear prediction.

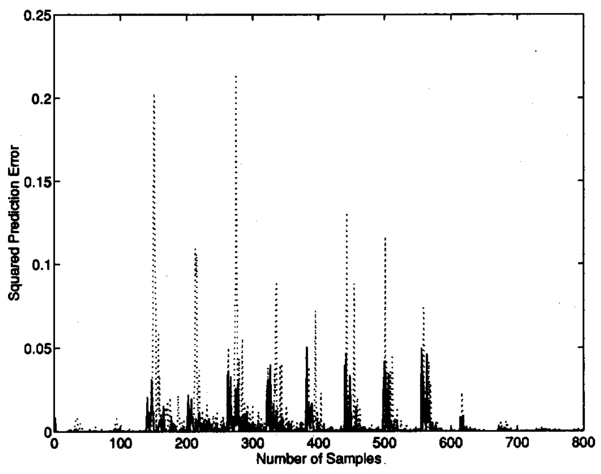

Fig. 10. Squared prediction error versus the number of speech samples. Continuous curve: nonlinear predictions. Dashed curve: linear prediction.

Fig. 7 shows the power spectrum of the resulting prediction error sequence. The power spectral density is fairly constant across a band from 200–3200 Hz. The solid line presents the average power spectrum at each frequency point, whereas the two dashed lines correspond to the 95% confidence range. Fig. 8 shows the histogram of the prediction error, which is found to have a β distribution. The conclusion to be drawn from these two latter figures is that the nonlinear prediction error may be closely modeled as a white and approximately Gaussian process, indicating that it consists essentially of statistically independent samples.

B. Performance Comparisons

Continuing with experimental results, Fig. 9 shows the prediction performed by a linear (tapped-delay-line) predictor with 12 taps, using the same speech signal as that used in Fig. 6. Comparing Figs. 6 and 9, we see that the prediction of a speech signal using a nonlinear adaptive predictor provides a much better approximation to the actual speech signal than the linear predictor. The improved performance of the nonlinear adaptive predictor compared with the linear adaptive predictor is more vividly displayed in Fig. 10; the solid curve in this figure corresponds to the squared nonlinear prediction error versus the number of samples, whereas the dashed curve corresponds to the case of linear prediction.

For a quantitative evaluation of prediction performance, we may use the following error measure expressed in decibels:

$$R_p = 10\log_{10}(\delta_s^2/\delta_p^2) \tag{16}$$

where δ_s^2 is the mean-square value of the incoming speech signal, and δ_p^2 is the corresponding value of the prediction error at the predictor output. For 10 000 speech samples, δ_s^2 is calculated to be about 0.3848. For the PRNN-based nonlinear predictor δ_p^2 is about 1.18×10^{-3}, yielding R_p = 25.14 dB. On the other hand, for the linear adaptive predictor, δ_p^2 is about

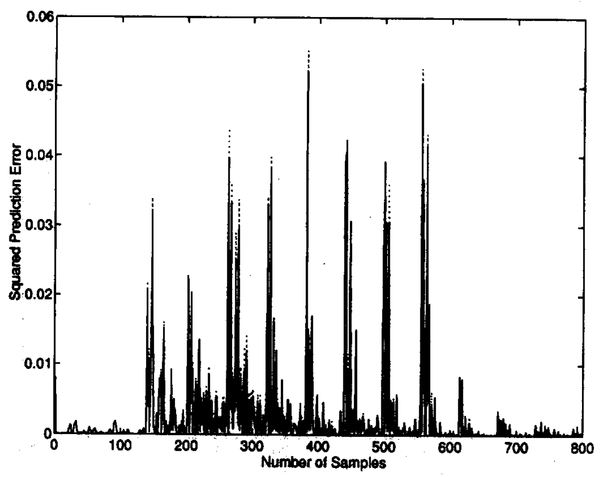

Fig. 11. Squared prediction error versus the number of speech samples. Solid curve: the PRNN-based nonlinear adaptive predictor. Dashed curve: nonlinear adaptive predictor using a conventional recurrent neural network.

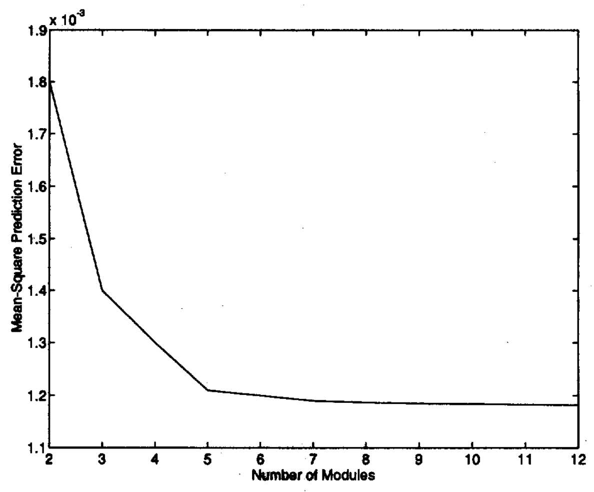

Fig. 12. Illustrating the relation between the mean-square prediction error and the number of modules for the case of a speech signal.

2.5×10^{-3}, yielding $R_p = 22.01$ dB. These results indicate a processing gain of 3.13 dB for the nonlinear predictor over the linear one.

Finally, Fig. 11 compares the performance of the PRNN-based nonlinear adaptive predictor with that of a corresponding predictor using a conventional (i.e., a single, fully connected) recurrent neural network trained with the RTRL algorithm. Both networks employ the same total number of neurons (i.e., 10) and the same tapped-delay line section (with 12 taps). The solid curve pertains to the squared prediction error plotted versus the number of speech samples for the PRNN case, whereas the dashed curve pertains to the nonlinear adaptive predictor using the conventional recurrent network. On the whole, the two nonlinear adaptive predictors exhibit a similar performance, although the PRNN-based structure appears to have a slight advantage. The main point to note, however, is that with five modules and two neurons per module, the computational complexity of training the PRNN is on the order of 5×2^4, whereas the computational complexity of training the conventional recurrent network is on the order of 104. For the application described here, the PRNN reduces the computational complexity by more than two orders of magnitude.

C. Choice of Design Parameters

The choice of values assigned to the number of modules M, the number of neurons per module N, and the number of taps q in the nonlinear adaptive filtering system (whixh were described in Figs. 3 and 4) were determined experimentally, as described here:

- In principle, the more modules we use, the more accurate is the "linearization" of the input space performed by the PRNN. Fig. 12 shows the relation between the mean-square prediction error of the speech signal and the number of modules M, assuming that $N = 2$, $p = 4$, and $q = 12$. From this figure, we see that for M larger than 6, the mean-square prediction error is almost constant.
- The linearization of the input space may also be improved by increasing the number of neurons per module N. Fig. 13 illustrates the effect of varying N on the mean-square prediction error, assuming that $M = 5$, $p = 4$, and $q = 12$. As expected, we see that increasing N has the effect of reducing the mean-square prediction error, but this is achieved in a very gradual manner. In any event, the use of a large N is not recommended since the computational requirement increases sharply with N, hence, the choice of $N = 2$ as the design value.
- Intuitively, we expect the accuracy of the one-step prediction performed by the linear subsection to be enhanced by increasing the number of taps q. This is borne out by the results shown in Fig. 14, where the mean-square prediction error is plotted versus q for $M = 5$, $N = 2$, and $p = 4$. As q increases from 2 to 12, the mean-square prediction error decreases sharply, and once q exceeds 12, the prediction error is reduced only by a small amount on the average. This result is in keeping with the order of about 12 for an autoregressive (AR) model of speech reported in [13].

Finally, the choice of the nonlinear prediction order $p = 4$ for the PRNN follows [14].

One other design parameter that needs to be specified is the number of pretraining samples N_0. In general, if N_0 is too small, the epochwise training method may not determine an adequate initial weight matrix for the recurrent neural network. An inadequate initial value for the weight matrix may cause the PRNN to diverge. The main purpose of pretraining is merely to determine an adequate set of initial weights. To economize on pretraining time, the size of pretraining samples N_0 is limited to about one or two hundreds. This choice is also justified experimentally as follows. Fig. 15 shows a plot of the mean-square prediction error versus N_0. Here, we see that when N_0 is less than 50, the mean-square prediction error rises very sharply, indicating divergence of the adaptive prediction process. When N_0 exceeds 50, the mean-square prediction

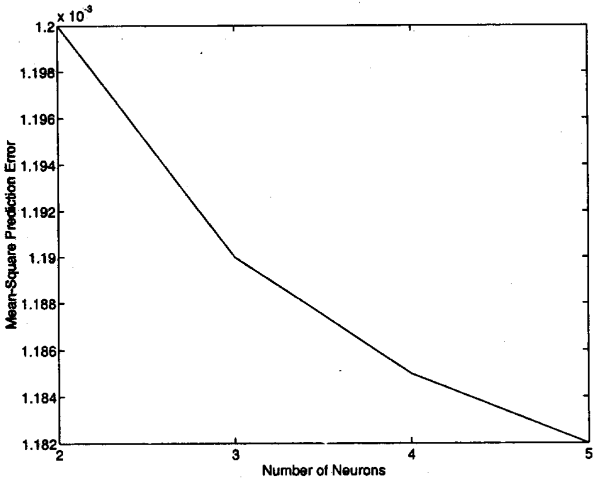

Fig. 13. Illustrating the relation between the mean-square prediction error and the number of neurons in each module for the case of a speech signal.

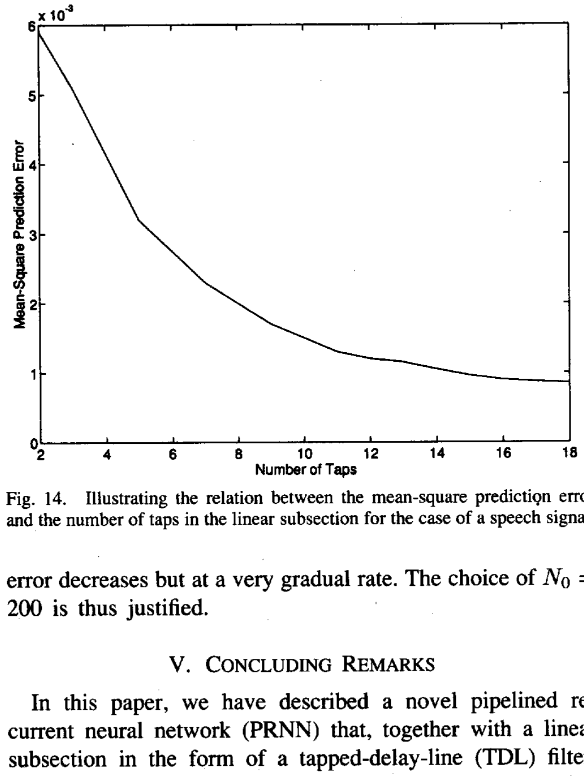

Fig. 14. Illustrating the relation between the mean-square prediction error and the number of taps in the linear subsection for the case of a speech signal.

error decreases but at a very gradual rate. The choice of $N_0 = 200$ is thus justified.

V. Concluding Remarks

In this paper, we have described a novel pipelined recurrent neural network (PRNN) that, together with a linear subsection in the form of a tapped-delay-line (TDL) filter, provides a powerful device for the nonlinear adaptive filtering of nonstationary time series. The adaptive signal processing capability of this new network has been demonstrated by studying the one-step prediction of speech signals. In addition to this capability, the new network has the potential for solving other difficult nonlinear adaptive signal processing tasks such as system identification, adaptive equalization, and adaptive noise cancellation where nonlinearity and nonstationarity are both important factors; the latter applications have yet to be explored.

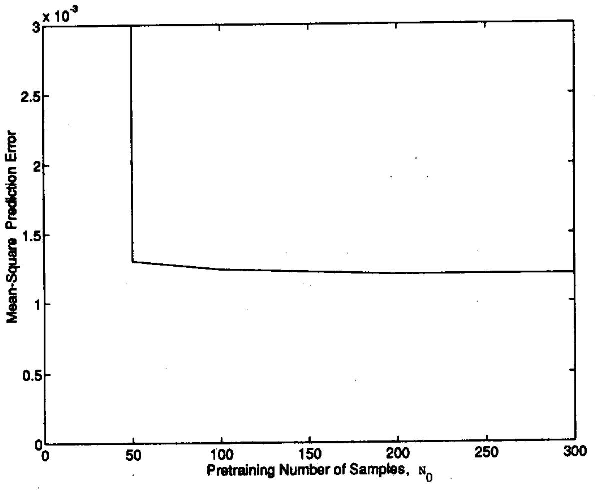

Fig. 15. Plot of the mean-square prediction error versus the number of pretraining samples.

An important attribute of the new neural network-based adaptive predictor is its high computational efficiency. Specifically, the total computational requirement of processing a single sample is $O(MN^4 + 3(q+1))$ arithmetic operations, where

M	number of modules
N	number of neurons per module in the PRNN
q	number of taps in the TDL filter.

For a total number of MN neurons in the PRNN, this is to be contrasted with the computational requirement of a corresponding structure involving the use of a conventional recurrent neural network, which is $O(M^4N^4 + 3(q+1))$ arithmetic operations. Thus, the computational savings made possible by the use of the PRNN can indeed be enormous for large M.

The new nonlinear adaptive predictor has been successfully applied to the adaptive differential pulse-code modulation (ADPCM) of speech signals [15], [16]. Computer simulation and listening tests on different speech signals show that the nonlinear ADPCM so designed continues to perform satisfactorily at bit rates as low as 16 kb/s.

Appendix
Algorithm for On-Line Training and Nonlinear Adaptive Prediction

1. Nonlinear Subsection

- **Prediction**

  At the nth time point, the input vectors $\boldsymbol{s}_1(n), \cdots, \boldsymbol{s}_M(n)$ are obtained, where

$$\boldsymbol{s}_i(n) = [s(n-i), s(n-i-1), \cdots, s(n-(i+p-1))]^T \quad (17)$$

The output signal of module i and the error signal of level i are defined by, respectively

$$y_{i,1}(n) = \phi(\boldsymbol{W}, \boldsymbol{s}_i(n), \boldsymbol{r}_i(n)) \quad (18)$$

$$e_i(n) = s(n-i+1) - y_{i,1}(n) \tag{19}$$

where $i = 1, 2, \cdots, M$, and $y_{i,1}(n)$ denotes the one-step prediction computed by the ith module, and $e_i(n)$ is the corresponding error signal; $\boldsymbol{W}$ is the synaptic weight matrix of module i, and $\boldsymbol{r}_i(n)$ is its feedback signal vector. After every module of PRNN finishes its calculations, a series of error signals are obtained, namely, $e_1(n)$, $e_2(n), \cdots, e_M(n)$.

- **Updating of the weight matrix $\boldsymbol{W}$**

  The overall cost function for the pipelined recurrent neural network (PRNN) is defined by

$$\mathcal{E}(n) = \sum_{i=1}^{M} \lambda^{i-1} e_i^2(n) \tag{20}$$

  where λ is an exponential forgetting factor that lies in the range $(0 < \lambda \le 1)$. The change applied to the klth element of the weight matrix $\boldsymbol{W}$ is

$$\Delta w_{k,l} = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{k,l}} \tag{21}$$

  where η is a fixed learning-rate parameter $1 \le k \le N$ and $1 \le l \le (p+N+1)$. Hence, using a modification of the RTRL algorithm [6], the change $\Delta \boldsymbol{W}$ applied to the weight matrix $\boldsymbol{W}$ using (21) is calculated, and the weight matrix is updated as

$$\boldsymbol{W}_{\text{new}} = (\boldsymbol{W} + \Delta \boldsymbol{W}) \tag{22}$$

- **Filtering**

  Using the updated weight matrix $\boldsymbol{W}_{\text{new}}$, and the updated p-by -1 input vectors with leading elements $s(n)$, $s(n-1), \cdots, s(n-M)$, the filtered signal $y_{\text{filt}}(n)$ at the output of the first visible neuron of module 1 in the PRNN is computed.

2. Linear Subsection

The one-step prediction computed by the TDL filter is

$$\hat{s}(n+1) = \boldsymbol{w}_l^T \boldsymbol{y}_{\text{filt}}(n) \tag{23}$$

where $y_{\text{filt}}(n)$ is derived from the PRNN:

$$\boldsymbol{y}_{\text{filt}}(n) = [y_{\text{filt}}(n), y_{\text{filt}}(n-1), \cdots, y_{\text{filt}}(n-q+1)]^T \tag{24}$$

The sample $s(n+1)$ of the original signal represents the desired response; hence, the error signal is defined by

$$e(n+1) = s(n+1) - \boldsymbol{w}_l^T \boldsymbol{y}_{\text{filt}}(n) \tag{25}$$

The weight vector of the TDL filter is updated in accordance with the LMS algorithm:

$$\boldsymbol{w}_l \leftarrow \boldsymbol{w}_l + \mu \boldsymbol{y}_{\text{filt}}(n) e(n+1) \tag{26}$$

where μ is the step-size parameter.

3. Recursive Calculation

Let $n = n+\ 1$ and return to step 1. Repeat the nonlinear adaptive prediction until the input signal is terminated.

Initialization of the Algorithm: To proceed with the computation, we need to initialize the tap-weight vector $\boldsymbol{w}_l$ of the TDL filter and synaptic weight matrix $\boldsymbol{W}$ of every module in the PRNN. We also need to specify the initial feedback signal vector $\boldsymbol{r}$ of very module.

Initialization of the tap-weight vector $\boldsymbol{w}_l$ follows the customary practice of setting it equal to the null vector or a randomly distributed set of small values. However, initialization of the synaptic weight matrix $\boldsymbol{W}$ and the feedback signal vector $\boldsymbol{r}$ require special attention. For this initialization, we may use the traditional epochwise training method of recurrent neural networks, which is applied simply to one module operating with N_0 samples of the input signal.

The training procedure is summarized as follows:

1) Input N_0 samples of the signal $s(t)$, and obtain a p-by -1 input vector $\boldsymbol{s}(i)$ and a desired signal $d(i)$, when $1 \le i \le n'$ and $n' = N_0 - p$:

$$\boldsymbol{s}(i) = [s(i+(p-1)), \cdots, s(i)]^T \tag{27}$$

$$d(i) = s(i+p) \tag{28}$$

2) The pretraining procedure beings with $i = 1$. Choose a random set of small values for the $(p+N+1)$-by $-N$ weight matrix $\boldsymbol{W}$ and the N-by -1 feedback vector $\boldsymbol{r}$ (1).
3) Input the $\boldsymbol{s}(i), \boldsymbol{r}(i)$, and $\boldsymbol{d}(i)$ to a module, and perform the following calculations:

$$y_k(i) = \phi(v_k(i)) \tag{29}$$

$$v_k(i) = \sum_{l=1}^{p} w_{k,l} s(i+(p-l) + w_{k,p+1} + \sum_{l=p+2}^{p+N+1} w_{k,l} r_{l-(p-1)}(i) \tag{30}$$

$$e(i) = s(i+p) - y_1(i) \tag{31}$$

where the weight $w_{k,p+1}$ represents the adjustable bias.

4) Let $i = i+1$, and set

$$\boldsymbol{r}(i) = \boldsymbol{y}(i-1) \tag{32}$$

and return to step 3.

5) Repeat the calculation of steps 3 and 4 until $i = n'$. If the cost function $E(n')$ is less than a permitted value $\mathcal{E}$, the pretraining procedure is stopped; otherwise, go to step 6. The cost function $E(n')$ is defined by

$$E(n') = \frac{1}{n'} \sum_{i=1}^{n'} e^2(i) \tag{33}$$

6) Compute the change $\Delta \boldsymbol{W}$ by using the gradient estimation algorithm along the negative of the gradient of

$E(n')$ with respect to $\boldsymbol{W}$, and update the weight matrix:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta\boldsymbol{W} \tag{34}$$

7) Set $i = 1$, and $\boldsymbol{r}(1) = \boldsymbol{y}(n')$ and return to step 3.

To economize on pretraining time, the number of pretraining samples N_0 is limited to about 100–200 samples. Roughly speaking, the permitted error e is about 1% of the mean-square value of the input signal $s(t)$.

ACKNOWLEDGMENT

Comments made by anonymous reviewers of an early version of the paper are appreciated.

REFERENCES

[1] S. Shuzo and K. Nakata, *Fundamentals of Speech Signal Processing.* New York: Academic, 1985.
[2] N. S. Jayant and P. Noll, *Digital Coding of Waveforms.* Englewood Cliffs, NJ: Prentice-Hall, 1984.
[3] D. Gabor, W. P. L. Wilby, and R. Woodcock, "An universal nonlinear filter, predictor and simulator which optimizes itself by a learning process," *Proc Inst. Elec. Eng.*, vol. 108, pp. 422–438, 1961.
[4] S. Haykin and L. Li, "Real-time nonlinear adaptive prediction of nonstationary signal," CRL Rep. No. 272, Commun. Res. Lab., McMaster Univ., Oct. 1993.
[5] L. Li and S. Haykin, "A cascaded recurrent neural network for real-time nonlinear adaptive filtering," in *Proc. International Conference on Neural Networks* (San Francisco), 1993, pp. 857–862.
[6] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, pp. 270–280, 1989.
[7] L. E. McBride, Jr. and K. S. Narendra, "Optimization of time-varying systems," *IEEE Trans. Automat. Contr.*, vol. 10, pp. 289–294, 1965.
[8] D. C. Van Essen, C. H. Anderson, and D. J. Felleman, "Information processing in the primate visual system: An integrated systems perspective," *Sci.*, pp. 419–423, Jan. 1992.
[9] J. C. Houk, "Learning in modular networks," in *Proc. Int. Workshop Adaptive Learning Syst.*, (New Haven, CT), May 1992, pp. 80–84.
[10] S. Haykin, *Neural Networks: A Comprehensive Foundation.* New York: Macmillan, 1994.
[11] ——, *Adaptive Filter Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1991, 2nd ed.
[12] B. Widrow and S. Stearns, *Adaptive Signal Processing*Englewood Cliffs, NJ: Prentice-Hall, 1985.
[13] B. S. Atal and S. L. Hanauer, "Speech analysis and synthesis by linear prediction of the speech wave," *J. Acoust. Soc. Amer.*, vol. 50, pp. 637–655, 1971.
[14] B. Townshend, "Nonlinear prediction of speech," in *Proc. Int. Conf. Acoust. Speech Signal Processing* (Toronto, Canada), 1991, pp. 425–427.
[15] S. Haykin and L. Li, "16 kb/s adaptive differential pulse code modulation of speech," in *Proc. Int. Workshop Applications Neural Networks Telecommun.* (Princeton, NJ), 1993, pp. 132–138.
[16] L. Li, "Nonlinear adaptive prediction of nonstationary signals and its application to speech communications," Ph.D. dissertation, Dept. of Elect. and Comput. Eng., McMaster Univ., 1994.

Simon Haykin (F'82) received the B.Sc. degree (with First-Class Honours) in 1953, the Ph.D. degree in 1956, and the D.Sc. degree in 1967, all in electrical engineering from the University of Birmingham, England.

He is the founding Director of the Communications Research Laboratory and Professor of Electrical and Computer Engineering at McMaster University, Hamilton, Canada. His research interests include nonlinear dynamics, neural networks, and adaptive filters and their applications. He is the editor of *Adaptive Learning Systems for Signal Processing, Communications, and Control*, which is a new series of books by Wiley Interscience

In 1980, Dr. Haykin was elected Fellow of the Royal Society of Canada. He was awarded the McNaughton Gold Medal, IEEE (Region 7), in 1986 and the Canadian Award in Telecommunications in 1992.

Liang Li was born in the People's Republic of China in 1961. He received the B.Eng. and M.Eng. degrees from the Beijing Institute of Technology in electrical engineering in 1983 and 1988, respectively.

Since 1990, he has been a Ph.D. candidate with the Communications Research Laboratory, McMaster University, Hamilton, Canada. He is currently a software engineer with Bell Northern Research Inc. His research interests include speech signal processing, neural networks, and communications.