Nonlinear Adaptive Prediction of Speech with a Pipelined Recurrent Neural Network

Jens Baltersee and Jonathon A. Chambers, *Member, IEEE*

***Abstract*—New learning algorithms for an adaptive nonlinear forward predictor that is based on a pipelined recurrent neural network (PRNN) are presented. A computationally efficient gradient descent (GD) learning algorithm, together with a novel extended recursive least squares (ERLS) learning algorithm, are proposed. Simulation studies based on three speech signals that have been made public and are available on the World Wide Web (WWW) are used to test the nonlinear predictor. The gradient descent algorithm is shown to yield poor performance in terms of prediction error gain, whereas consistently improved results are achieved with the ERLS algorithm. The merit of the nonlinear predictor structure is confirmed by yielding approximately 2 dB higher prediction gain than a linear structure predictor that employs the conventional recursive least squares (RLS) algorithm.**

***Index Terms*—Adaptive algorithms, neural networks, nonlinear prediction.**

I. Introduction

MANY SIGNALS are generated from an inherently nonlinear physical mechanism and have statistically nonstationary properties, a classic example of which is the speech signal. Linear structure adaptive architectures are suitable for the prediction of such signals, but they do not exploit their inherent nonlinearity and associated higher order statistics. Adaptive techniques that account for the nonlinear nature of the signal should therefore outperform conventional linear adaptive techniques. An emergent, nonlinear structure suitable for prediction is the artificial neural network (ANN). In 1995, Haykin and Li [2] presented a novel, computationally efficient nonlinear predictor based on a pipelined recurrent neural network (PRNN). The learning algorithm used by Haykin and Li for the PRNN is a gradient descent algorithm. This paper presents new learning algorithms for the nonlinear predictor, namely, a computationally more efficient gradient descent (GD) learning algorithm and a novel extended recursive least squares (ERLS) learning algorithm. Simulations, based on three speech signals available from the author's WWW home page [7] are used to test the nonlinear predictor and the appropriate new learning algorithms.

Manuscript received July 8, 1996; revised February 19, 1998. The associate editor coordinating the review of this paper and approving it for publication was Dr. Shigeru Katagiri.

J. Baltersee is with the Integrated Systems for Signal Processing Laboratory, Aachen University of Technology (RWTH), Aachen, Germany (e-mail: balterse@ert.rwth-aachen.de).

J. A. Chambers is with the Signal Processing Section, Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, London, U.K. (e-mail: j.chambers@ic.ac.uk).

Publisher Item Identifier S 1053-587X(98)05230-1.

The paper is organized in the following manner. In Section II, the nonlinear predictor, as presented by Haykin and Li in [2], is described; it consists of a PRNN trained by a GD learning algorithm and a linear LMS postprocessor. It is shown in Section III how this GD learning algorithm can be modified in order to reduce computational complexity without affecting the prediction performance. Furthermore, in Section IV, a novel learning algorithm for the PRNN based on the ERLS algorithm is proposed. In Section V, the performance of the nonlinear predictor using both the GD and the ERLS algorithm is compared with the performances of conventional linear structure predictors using the least mean square (LMS) and recursive least squares (RLS) algorithms. The simulation results indicate that the GD algorithm, which is also used by Haykin and Li in [2], is not the best learning algorithm for the PRNN. Improved results are obtained with the newly introduced ERLS learning algorithm. Section VI summaries the main results and concludes the paper.

II. The Nonlinear Adaptive Predictor

The nonlinear predictor proposed by Haykin and Li [2] is based on the notion of first linearizing the input signal with the help of the PRNN and then feeding these processed data into a conventional linear predictor to yield a one-step forward prediction of the original signal. This combination of nonlinear and linear processing should be able to extract both nonlinear and linear relationships contained in the input signal. It is expected that such a predictor will outperform, in terms of prediction gain as defined later, a conventional linear predictor when applied to signals generated by some nonlinear underlying mechanism. Fig. 1 represents the prediction process due to Haykin and Li.

A. The Pipelined Recurrent Neural Network

The linearization of the input signal is achieved using a PRNN. A PRNN consists of a number of small-scale recurrent neural network (RNN) modules, which is shown in Fig. 2. In the processing layer, the inputs are linearly combined and then fed through a nonlinear activation function $\Phi(\cdot)$ to form the output of each neuron. The weights of each neuron k form a weight vector $\mathbf{w}_k^T = [w_{k,1}, \cdots, w_{k,p+F}]$, where p is the number of external inputs, and F is the number of feedback connections. The weight vectors from each neuron form a weight matrix $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_N]$, with N being the number of neurons in the network. For more details about recurrent neural networks and their learning algorithms, refer

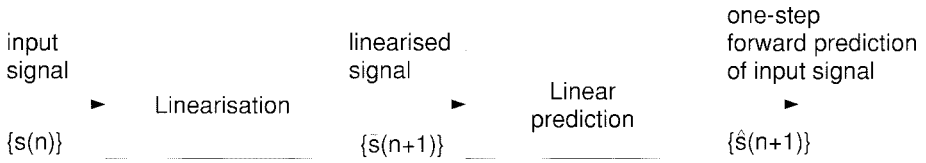

Fig. 1. Two-step prediction process of the nonlinear predictor.

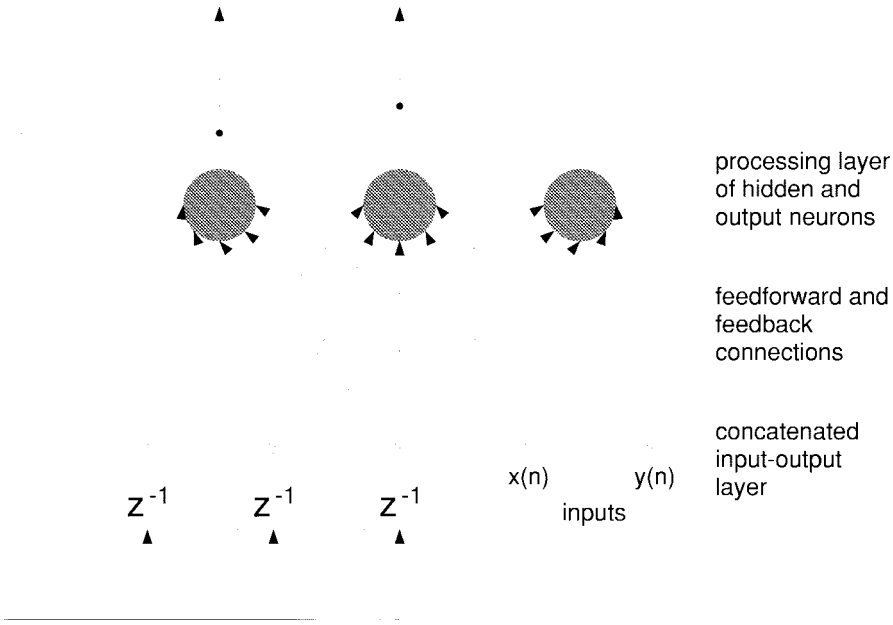

Fig. 2. Fully connected recurrent neural network.

to the comprehensive paper by Williams and Zipser [5]. The network shown in Fig. 2 is a special type of recurrent neural network since all of its outputs, whether hidden or not, are fed back to the input layer of the network, i.e., $F = N$ in this case. Such a network is called a fully connected recurrent neural network. In the PRNN, M recurrent neural networks are connected as shown in Fig. 3. Module M of the PRNN is a fully connected RNN, whereas in modules $\{M-1, \cdots, 1\}$, one of the feedback signals is substituted with the output of the first neuron of the following module. The $p \times 1$ external signal vector $\mathbf{s}^T(n) = [s(n-1), \cdots, s(n-p)]$ is delayed by $z^{-m}\mathbf{I}$ at the input of the module m, where z^{-m} denotes the delay operator of m time units, and $\mathbf{I}$ is the identity matrix. All of the modules have the same $(N+p+1) \times N$ weight matrix $\mathbf{W}$, which is time variant, i.e., $\mathbf{W} = \mathbf{W}(n)$. The overall output of the PRNN is the output of the first neuron of the first module, i.e., $y_{1,1}$, as shown in Fig. 3. The PRNN, as presented in [2], employs a learning algorithm that is based on the GD algorithm, as in [5]. This kind of GD algorithm is called a real time recurrent learning (RTRL) algorithm, which is particularly suitable for the prediction of nonstationary (e.g., speech) signals since the weights are continuously adapted. Continuous adaptation means that at every discrete time step n, a correction $\Delta\mathbf{W}(n)$ to the weight matrix of a module $\mathbf{W}(n)$ is calculated, which is added to $\mathbf{W}(n)$ in order obtain the updated weight matrix $\mathbf{W}(n+1)$. The adaptation itself is an optimization technique of the stochastic gradient (SG) type. Note that this is **not** the case for most other neural network learning algorithms, which usually fix the weights after an initial training period and are thus not suitable for prediction of nonstationary signals. The merit of a PRNN, as compared with a single fully connected recurrent neural network, is that its computational complexity is reduced for the same total number of neurons. Let the total number of neurons in an RNN be N. If M RNN's, each with N neurons, constitute the modules of the PRNN, then the total number of neurons in the PRNN is $M{\cdot}N$. The computational complexity of a fully connected RNN consisting of the same number of neurons, trained with a GD type algorithm, say the RTRL algorithm, would be $\mathcal{O}((M{\cdot}N)^4)$ [2], [5]. The PRNN architecture achieves a reduction in the computational complexity for the same number of neurons to $\mathcal{O}(M{\cdot}N^4)$ [2]. Another advantage of the PRNN over a single RNN is its increased inherent nonlinearity, which results from the cascading of several nonlinear modules.

Equations (1) give a full description of the dynamics of the PRNN, as depicted in Fig. 3

$$
\begin{aligned}
y_{i,k}(n) &= \Phi(v_{i,k}(n)) \\
v_{i,k}(n) &= \sum_{l=1}^{p+N+1} w_{k,l}(n) u_{i,l}(n) \\
\mathbf{u}_i^T(n) &= [s(n-i), \cdots, s(n-i-p+1), 1 \\
&\qquad y_{i+1,1}(n), y_{i,2}(n-1), \cdots, y_{i,N}(n-1)] \\
&\qquad \text{for} \quad 1 \le i \le M-1 \\
\mathbf{u}_i^T(n) &= [s(n-i), \cdots, s(n-i-p+1), 1 \\
&\qquad y_{i,1}(n-1), y_{i,2}(n-1), \cdots, y_{i,N}(n-1)] \\
&\qquad \text{for} \quad i = M
\end{aligned} \tag{1}
$$

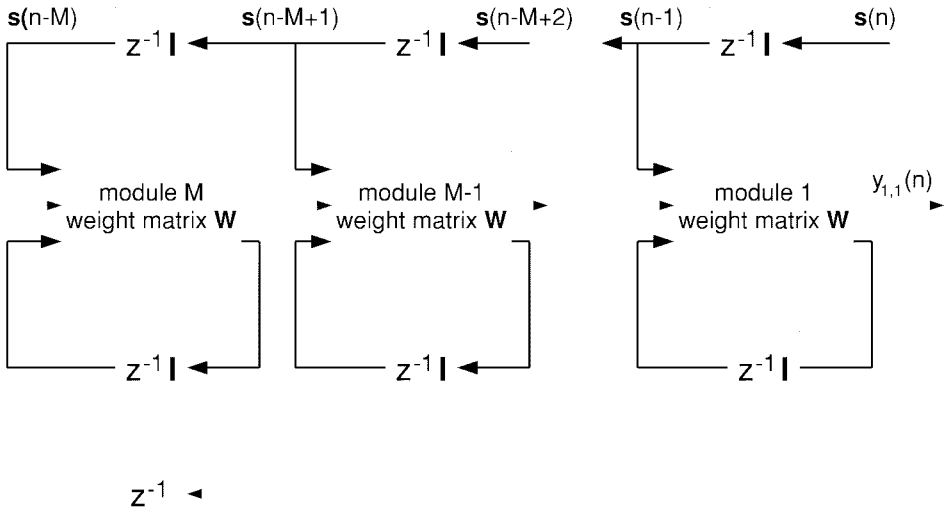

Fig. 3. Pipelined recurrent neural network.

where

- $y_{i,k}(n)$ output of the kth neuron in module i;
- $v_{i,k}(n)$ internal activity (net input) of the kth neuron in module i;
- $w_{k,l}$ weight for the lth input to the kth neuron within a module;
- $\Phi(\cdot)$ nonlinear activation function for all of the neurons in the network;
- p number of external inputs to a module in the PRNN;
- $(.)^T$ vector transpose.

Given the input vectors $\mathbf{u}_i(n) \subset \{s(n)\}$ for each module i $(i = 1, \cdots, M)$, the outputs of all neurons in the network at the time step n can be calculated (1). The output of a module i is defined as the output of its first neuron $y_{i,1}$. As the particular modules in the PRNN configuration are connected, the overall output of the PRNN becomes the output of the first neuron of module 1, i.e., $y_{1,1}$. The one-step prediction error of module i at time step n, which is denoted by $e_i(n)$, is then defined as the difference between the desired response at that module at the time step n, i.e., $s(n-i+1)$ and the output of the module i at the time step n, which is denoted by $y_{i,1}(n)$

$$e_i(n) = s(n-i+1) - y_{i,1}(n). \tag{2}$$

In order to implement the PRNN, it is necessary to derive a learning algorithm based on the gradient of the cost function of the PRNN, which is a measure of the sum of squared instantaneous errors over all of the outputs of PRNN modules (4) at the time instant n. The learning algorithm must calculate the correction term $\Delta\mathbf{W}(n)$ in order to update the weight matrix $\mathbf{W}(n)$ at each discrete time step n. The continuous adaptation of the weight matrix is necessary since the predictor operates on the nonstationary data. Note that having the same weight matrix $\mathbf{W}(n)$ for all modules makes the task of finding such a learning algorithm considerably easier. The values of the elements of the weight matrix $\mathbf{W}$ at time step $(n+1)$ can be calculated using

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \Delta\mathbf{W}(n) \tag{3}$$

where $\mathbf{W}(n)$ is an $(N+p+1) \times N$ matrix, with N columns having $(N+p+1)$ elements representing the weights belonging to each of the N neurons and having an additional weight for the constant bias input, which is included in the input vector.

Two learning algorithms are derived in the sequel: a GD learning algorithm and a novel algorithm based on the ERLS algorithm. Both learning algorithms should minimize an overall error function of the PRNN, as given in (4). Since the PRNN consists of M modules, there are M instantaneous error signals. Both learning algorithms must minimize a **sum** of these M squared instantaneous error signals. A forgetting factor λ, which determines the weighting of the individual modules, is introduced. Thus, the error function of the PRNN over all modules at the time instant n becomes the cost function $E(n)$ of the PRNN given as

$$E(n) = \sum_{i=1}^{M} \lambda^{i-1} e_i^2(n) \tag{4}$$

where $e_i(n)$ is given in (2).

B. The Linearization of the Input Signal

The first operation of the nonlinear predictor is the linearization of the input signal $\{s(n)\}$ using a PRNN, as shown in Fig. 1. The linearization procedure itself consists of three steps:

- *Prediction:* Compute the one-step nonlinear forward prediction errors of the PRNN at the discrete time instant n (2), and calculate (4).
- *Weight Updating:* A learning algorithm uses the gradient of the cost function (4) to calculate the weight matrix correction $\Delta\mathbf{W}(n)$, which is added to the weight matrix

of a module $\mathbf{W}(n)$ in order to build the updated weight matrix $\mathbf{W}(n+1)$.

- *Filtering:* The output signal of the PRNN at the next time instant $(n+1)$ is computed (1). This is performed by recalculating all the outputs $y_{i,k}(n+1)(i=1,\cdots,M,\ k=1,\cdots,N)$ of the modules using the updated weight matrix of a module $\mathbf{W}(n+1)$ and the updated input vectors to the modules $\mathbf{u}_i^T(n+1)$. The updated input vectors are formed by substituting the external input vector $\mathbf{s}_i(n)=[s(n-i),\cdots,s(n-i-p+1)]$ with $\mathbf{s}_i(n+1)$ in the input vectors $\mathbf{u}_i(n)$ for each module $i=1,\cdots,M$. In Haykin's and Li's nonlinear predictor [2], the feedback values contained in the input vectors $\mathbf{u}_i(n)$ were **not** updated at this stage. Later on, in Section III, it will be shown that it is advantageous to update the feedback values at this step, as it reduces the computational complexity of the nonlinear predictor. The feedback values at the time step $(n+1)$ consist, therefore, of the filtered outputs at time step n.

C. The Linear Subsection

The output signal of the nonlinear subsection $\{\overline{s}(n+1)\}$ is finally fed into the linear subsection of the nonlinear predictor in order to accomplish the one-step forward prediction process. The linear subsection is a conventional linear predictor, using either an LMS or an RLS algorithm to update the adjustable weights. For an introduction to linear prediction, refer to [1]. The nonlinear activation function $\Phi(\cdot)$ used in the PRNN is the logistic sigmoid function whose amplitude lies in $[0,\ 1)$, which makes the output of the PRNN nonzero mean. It is well known that a constant bias input should be included among the inputs to a linear filter so that the LMS or RLS algorithm works well on nonzero mean input signals. This fact was ignored in Haykin and Li's paper. If the bias is included, then the tap length of the linear predictor was chosen as 12, which is standard for telephonic quality speech, $+1$ for the constant bias input.

III. A GD Learning Algorithm and a Modified Nonlinear Predictor

A GD algorithm can be derived for the PRNN following the approach of [5]. The idea is to calculate the correction $\Delta\mathbf{W}(n)$ to the weight matrix $\mathbf{W}(n)$ in the direction of the negative of the gradient of the cost function $E(n)$ (4). Hence, the change for the lth weight of neuron k at the time step n, i.e., $\Delta w_{k,l}(n)$, can be found as

$$\begin{aligned}\Delta w_{k,l}(n) &= -\eta\frac{\partial}{\partial w_{k,l}(n)}\left(\sum_{i=1}^{M}\lambda^{i-1}e_i^2(n)\right)\\ &= -2\eta\sum_{i=1}^{M}\lambda^{i-1}e_i(n)\frac{\partial e_i(n)}{\partial w_{k,l}(n)}.\end{aligned}\tag{5}$$

As the external input signal vector $\mathbf{s}$ does not depend on the elements of $\mathbf{W}$ and the errors $e_i(n)$ $(i=1,\cdots,M)$ at the output of the modules of the PRNN are calculated with respect to the output of the first neuron of the ith module $y_{i,1}(n)$, the partial derivative of the instantaneous error of the ith module in the PRNN, with respect to the weight $w_{k,l}$, i.e., $\partial e_i(n)/\partial w_{k,l}(n)$, becomes

$$\frac{\partial e_i(n)}{\partial w_{k,l}(n)}=-\frac{\partial y_{i,1}(n)}{\partial w_{k,l}(n)}.\tag{6}$$

Using the chain rule, the last equation becomes

$$\frac{\partial y_{i,1}(n)}{\partial w_{k,l}(n)}=\frac{\partial y_{i,1}(n)}{\partial v_{i,1}(n)}\frac{\partial v_{i,1}(n)}{\partial w_{k,l}(n)}=\dot{\Phi}(v_{i,1}(n))\frac{\partial v_{i,1}(n)}{\partial w_{k,l}(n)}\tag{7}$$

where $\dot{\Phi}$ denotes the first derivative of the activation function Φ with respect to its argument, which is the internal activity of the first neuron in the ith module $v_{i,1}$. Substituting (1) into (7) yields

$$\begin{aligned}&\dot{\Phi}(v_{i,1}(n))\frac{\partial v_{i,1}(n)}{\partial w_{k,l}(n)}\\ &\quad=\dot{\Phi}(v_{i,1}(n))\left(\sum_{\alpha=1}^{p+N+1}\left(\frac{\partial w_{1,\alpha}(n)}{\partial w_{k,l}(n)}u_{i,\alpha}(n)\right.\right.\\ &\qquad\left.\left.+\frac{\partial u_{i,\alpha}(n)}{\partial w_{k,l}(n)}w_{1,\alpha}(n)\right)\right).\end{aligned}\tag{8}$$

The first term in the previous sum is zero except for $k=1$ and $l=\alpha$, and the only elements of the input vector $\mathbf{u}$ that depend on the elements of $\mathbf{W}$ are the feedback values in the input vector to the module, which are denoted by $\mathbf{r}$ (12). Now, (8) becomes

$$\begin{aligned}&\dot{\Phi}(v_{i,1}(n))\frac{\partial v_{i,1}(n)}{\partial w_{k,l}(n)}\\ &\quad=\dot{\Phi}(v_{i,1}(n))\left(\sum_{\alpha=1}^{N}\frac{\partial r_{i,\alpha}(n)}{\partial w_{k,l}(n)}w_{1,\alpha+p+1}(n)+\delta_{k1}u_{i,l}(n)\right)\end{aligned}\tag{9}$$

where

$$\delta_{kl}=\begin{cases}1, & k=l\\ 0, & k\neq l.\end{cases}\tag{10}$$

If the feedback values in $\mathbf{u}_i(n)$ are also updated before the filtering part of the linearization procedure (see the previous section), then $\mathbf{r}_i(n)$ is defined by

$$\begin{aligned}\mathbf{r}_i^T=&[y_{i+1,1}(n),\ y_{i,2}(n-1),\cdots,y_{i,N}(n-1)]\\ &1\le i\le M-1\end{aligned}\tag{11}$$

$$\begin{aligned}\mathbf{r}_i^T=&[y_{i,1}(n-1),\ y_{i,2}(n-1),\cdots,y_{i,N}(n-1)]\\ &i=M.\end{aligned}\tag{12}$$

If the feedback values in $\mathbf{u}_i(n)$ are not updated, then the time index n in the previous equation has to be replaced with $(n-1)$ and $(n-1)$ with $(n-2)$. The effect of *not* updating the feedback values, as in [2], will be discussed at the end of this section. For the time being, it is assumed that the feedback

values are updated. Having defined the vector $\mathbf{r}_i^T$, it is now possible to formulate a recursive relationship for the partial derivatives of the elements of the feedback vector $r_{i,j}$ with respect to the weights $w_{k,l}$, i.e., $\partial r_{i,j}(n)/\partial w_{k,l}(n)$. For the case when ($1 \leq i \leq M$ and $2 \leq j \leq N$), (7) becomes

$$\begin{aligned}&\frac{\partial r_{i,j}(n)}{\partial w_{k,l}(n)}\\&\quad=\frac{\partial y_{i,j}(n-1)}{\partial w_{k,l}(n)}\\&\quad=\dot{\Phi}(v_{i,j}(n-1))\\&\qquad\cdot\left(\sum_{\alpha=1}^{N}\frac{\partial r_{i,\alpha}(n-1)}{\partial w_{k,l}(n)}w_{j,\alpha+p+1}(n-1)+\delta_{kj}u_{i,l}(n-1)\right).\end{aligned}\tag{13}$$

The partial derivatives on the right-hand side of (13) arise because the modules of the PRNN have feedback, whereby previous values of the output samples depend on the previous values of the weights, which, in turn, are related to the current weights via the weight updates algorithm. It should be noticed that these derivatives are made with respect to the present elements of $\mathbf{W}$ so that the expression (13) is no longer recursive. However, if the learning rate η for the PRNN is chosen sufficiently small so that the weights are supposed to adapt slowly, then under the approximation that $w_{k,l}(n+1)\approx w_{k,l}(n)$, (13) can be approximated by

$$\begin{aligned}\frac{\partial r_{i,j}(n)}{\partial w_{k,l}(n)}\approx\dot{\Phi}(v_{i,j}(n-1))&\left(\sum_{\alpha=1}^{N}\frac{\partial r_{i,\alpha}(n-1)}{\partial w_{k,l}(n-1)}\right.\\&\left.\cdot\, w_{j,\alpha+p+1}(n-1)+\delta_{kj}u_{i,l}(n-1)\right).\end{aligned}\tag{14}$$

For the case when ($1 \leq i \leq M-1$ and $j = 1$), no approximation needs to be made, and the upper relation becomes

$$\begin{aligned}&\frac{\partial r_{i,1}(n)}{\partial w_{k,l}(n)}\\&\quad=\frac{\partial y_{i+1,1}(n)}{\partial w_{k,l}(n)}\\&\quad=\dot{\Phi}(v_{i+1,1}(n))\\&\qquad\cdot\left(\sum_{\alpha=1}^{N}\frac{\partial r_{i+1,\alpha}(n)}{\partial w_{k,l}(n)}w_{1,\alpha+p+1}(n)+\delta_{kj}u_{i+1,l}(n)\right).\end{aligned}\tag{15}$$

Using (9), (14), and (15), the gradients of the outputs of the neurons can be expressed through the recursive relations

$$\begin{aligned}i=M\Rightarrow&\frac{\partial y_{i,j}(n)}{\partial w_{k,l}(n)}\\&\approx\dot{\Phi}(v_{i,j}(n))\left[\sum_{\alpha=1}^{N}\frac{\partial y_{i,\alpha}(n-1)}{\partial w_{k,l}(n-1)}w_{j,\alpha+p+1}(n)\right.\\&\qquad\left.+\delta_{kj}u_{i,l}(n)\right]\end{aligned}\tag{16}$$

$$\begin{aligned}i\neq M\Rightarrow&\frac{\partial y_{i,j}(n)}{\partial w_{k,l}(n)}\\&\approx\dot{\Phi}(v_{i,j}(n))\left[\frac{\partial y_{i+1,j}(n)}{\partial w_{k,l}(n)}w_{j,p+2}(n)+\sum_{\alpha=2}^{N}\right.\\&\qquad\cdot\frac{\partial y_{i,\alpha}(n-1)}{\partial w_{k,l}(n-1)}w_{j,\alpha+p+1}(n)\\&\qquad\left.+\delta_{kj}u_{i,l}(n)\right].\end{aligned}\tag{17}$$

Examining the derivation of the GD learning algorithm, it is noted that *not* updating the feedback values that are contained in the input vectors $\mathbf{u}_i(n)$ has a negative side effect; the gradient for the elements of the feedback vector does not equal the gradient for the outputs of the neurons

$$\frac{\partial r_{i,j}(n)}{\partial w_{k,l}(n)}\neq\frac{\partial y_{i,j}(n-1)}{\partial w_{k,l}(n)}.\tag{18}$$

Therefore, the recursions of (16) and (17) are not appropriate and should not be used to calculate the gradients of the neuron outputs. It is therefore necessary to first use the recursions of (14) and (15) in order to calculate the gradients of the feedback values. Once these gradients have been obtained, (9) can be used to calculate the gradients at the output neurons. This additional, triply indexed relation, with computational complexity of $O(M{\cdot}N^3)$, is not needed if the feedback values are updated before the start of the filtering part of the linearization procedure. Hence, by updating the feedback values, the total computational complexity of the GD learning algorithm is reduced. The prediction performance of the modified version is not affected by this change, as shown in Section V. Furthermore, the linearization procedure now consists of only two steps instead of three.

- *Prediction:* Compute the one-step prediction errors (2) from the PRNN, and calculate the cost function (4).
- *Weight Updating:* Use the SG algorithm on the cost function (4) to calculate the correction $\Delta\mathbf{W}(n)$, which is added to the weight matrix $\mathbf{W}(n)$ to form the updated weight matrix $\mathbf{W}(n+1)$.

There is no need for the explicit filtering procedure because the prediction part of the above linearization procedure at time step $(n+1)$ now comprises the filtering part of the linearization procedure at time step n.

IV. A Novel Learning Algorithm for the PRNN Based on the Extended Recursive Least Squares (ERLS) Algorithm

The novel extended recursive least squares (ERLS) algorithm, which is presented here, is derived from the extended Kalman filter (EKF) algorithm. For a thorough analysis of the EKF, see [3] and [4]. The cost function of the PRNN now becomes

$$\varepsilon(n)=\sum_{k=1}^{n}\xi^{n-k}E(k)\tag{19}$$

which must be minimized with respect to the weight matrix $\mathbf{W}$. The constant $\xi \in (0, 1]$ represents a forgetting factor that is introduced into the cost function (19) according to the RLS strategy and makes the resulting learning algorithm suitable for prediction of nonstationary signals. The ERLS algorithm is devised to solve the nonlinear minimization problem of (19) with respect to $\mathbf{W}$. In order to derive this algorithm, the starting point is the EKF system model and measurement model equations [4]

$$\begin{aligned}\mathbf{w}(n) &= \mathbf{a}(\mathbf{w}(n-1)) + \mathbf{u}(n)\\ \mathbf{x}(n) &= \mathbf{h}(\mathbf{w}(n)) + \mathbf{v}(n)\end{aligned} \tag{20}$$

where

- $\mathbf{w}(n)$ $N(N+p+1)\times 1$ weight vector obtained by rearranging the weight matrix $\mathbf{W}$ into a column vector $\mathbf{w}$ in a column-by-column manner;
- $\mathbf{x}(n)$ $M \times 1$ observation vector;
- $\mathbf{u}(n)$ vector of white Gaussian noise (WGN) $\mathbf{u}(n) \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$;
- $\mathbf{v}(n)$ observation noise that is a vector of WGN $\mathbf{v}(n) \sim \mathcal{N}(\mathbf{0}, \mathbf{C}(n))$ [4].

Furthermore, the functions $\mathbf{a}(\cdot)$ and $\mathbf{h}(\cdot)$ are assumed to be nonlinear, differentiable, and possibly time varying and are assumed to perform a mapping between Euclidean vector spaces

$$\mathbf{a} : \mathbf{R}^{N(N+p+1)} \rightarrow \mathbf{R}^{N(N+p+1)} \tag{21}$$

and

$$\mathbf{h} : \mathbf{R}^{N(N+p+1)} \rightarrow \mathbf{R}^{M}. \tag{22}$$

For the prediction of speech, the function $\mathbf{a}(\cdot)$ in (20) is unknown and may be approximated by the random walk model

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{u}(n) \tag{23}$$

whereas the nonlinear mapping function $\mathbf{h}(\cdot)$ can be linearized using a first-order Taylor expansion about the estimate $E(\mathbf{w}(n))$ based on its previous value, i.e., $E(\mathbf{w}(\mathbf{n})) = \hat{\mathbf{w}}(n|n-1)$, which yields

$$\begin{aligned}\mathbf{h}(\mathbf{w}(n)) \approx &\mathbf{h}(\hat{\mathbf{w}}(n|n-1)) + \nabla\mathbf{h}^T(\hat{\mathbf{w}}(n|n-1))\\ &\cdot [\mathbf{w}(n) - \hat{\mathbf{w}}(n|n-1)]\end{aligned} \tag{24}$$

where the gradient of $\mathbf{h}(\cdot)$ can be expressed as

$$\nabla\mathbf{h}^T = \frac{\partial \mathbf{h}(\hat{\mathbf{w}}(n|n-1))}{\partial \hat{\mathbf{w}}(n|n-1)} = \mathbf{H}(n) \tag{25}$$

so that the observation equation of (20) becomes

$$\begin{aligned}\mathbf{x}(n) = &\mathbf{H}(n)\mathbf{w}(n) + \mathbf{v}(n)\\ &+ [\mathbf{h}(\hat{\mathbf{w}}(n|n-1)) - \mathbf{H}(n)\hat{\mathbf{w}}(n|n-1)].\end{aligned} \tag{26}$$

Moreover, the correlation matrix of the state noise vector $\mathbf{u}(n)$ is equal to a scaled version of the minimum mean square error (MMSE) matrix of the EKF [3], [4]

$$\mathbf{Q}(n) = E\{\mathbf{u}(n)\mathbf{u}^T(n)\} = (\xi^{-1} - 1)\mathbf{M}(n) \tag{27}$$

where ξ is the forgetting factor introduced in (19). Using (23), (26), and (27), together with the EKF approach as in [4], the following equations of the ERLS algorithm can be obtained:

$$\begin{aligned}\mathbf{K}(n) = &\xi^{-1}\mathbf{M}(n-1)\mathbf{H}^T(n)\\ &\cdot [\mathbf{C}(n) + \xi^{-1}\mathbf{H}(n)\mathbf{M}(n-1)\mathbf{H}^T(n)]^{-1}\end{aligned} \tag{28}$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{K}(n)[\mathbf{x}(n) - \mathbf{h}(\hat{\mathbf{w}}(n-1))] \tag{29}$$

$$\mathbf{M}(n) = \xi^{-1}[\mathbf{I} - \mathbf{K}(n)\mathbf{H}(n)]\mathbf{M}(n-1). \tag{30}$$

For the PRNN, the $M \times 1$ vector $\mathbf{x}(n)$ becomes

$$\mathbf{x}^T(n) = [s(n),\, s(n-1),\, \cdots,\, s(n-M+1)]. \tag{31}$$

Furthermore, the vector $\mathbf{h}(n)$, which is the result of the nonlinear mapping $\mathbf{h}(n) = \mathbf{h}(\mathbf{w}(n))$ is actually the $M \times 1$ vector of the outputs of the PRNN modules

$$\mathbf{h}^T(n) = [y_{1,1}(n),\, y_{2,1}(n),\, \cdots,\, y_{M,1}(n)]. \tag{32}$$

Finally, using (23) for $E(\mathbf{w}(n))$, i.e., $\hat{\mathbf{w}}(n|n-1) = \hat{\mathbf{w}}(n-1|n-1) = \hat{\mathbf{w}}(n-1)$, the matrix $\mathbf{H}(n)$ of instantaneous gradients becomes

$$\mathbf{H}(n) = \frac{\partial \mathbf{h}(\hat{\mathbf{w}}(n-1))}{\partial \hat{\mathbf{w}}(n-1)} \tag{33}$$

whose elements are given in (16) and (17), which completes the derivation of the ERLS algorithm for the PRNN. The observation noise covariance matrix $\mathbf{Q}(n)$ of the ERLS algorithm was set to $c\mathbf{I}$ during the simulations $c \in \mathcal{R}^+$.

V. Experimental Results

The focus of this section is to assess the performance of the nonlinear predictor through simulations on three speech signals, which are denoted $s1$, $s2$, and $s3$. The signals are available from the author's WWW home page [7]. Signal $s2$ is identical to that used by Haykin and Li [2]. The measure that was used to assess the performance of the predictor is the one-step forward prediction gain R_p defined as

$$R_p \hat{=} 10 \log_{10}\left(\frac{\hat{\sigma}_s^2}{\hat{\sigma}_e^2}\right) \text{ dB} \tag{34}$$

where $\hat{\sigma}_s^2$ denotes the estimated variance of the input speech signal $\{s(n)\}$, and $\hat{\sigma}_e^2$ denotes the estimated variance of the error signal $\{e(n)\}$. This definition of the prediction gain is different from the one used in Haykin's and Li's paper, which uses the corresponding mean–squared values instead of the variance estimates. The estimates of variances have been chosen since the DC term contained in the mean squared values gives biased results.

The configurations of the nonlinear predictor that were tested are

- PRNN with GD + LMS in the linear subsection;
- PRNN with GD + RLS in the linear subsection;
- PRNN with ERLS + RLS in the linear subsection.

Each configuration was compared against the performance of a conventional linear predictor based on either the LMS or the RLS algorithm.

The initialization of the network weights was achieved via epochwise training, as is commonplace for neural networks with fixed weights. The elements of the initial weight matrix $\mathbf{W}(0)$ were chosen as small random numbers. The first L samples of the signal to be predicted were taken as the pretraining sequence. For L input samples, the appropriate weight corrections $\Delta\mathbf{W}$ were calculated after each input vector was presented to the network. These weight corrections were not added to $\mathbf{W}$ at each time step but were summed up as $\Delta\mathbf{W}_{epoch}$ and added to the weight matrix $\mathbf{W}$ at the very end of every epoch. Usually, a number of epochs is required to initialize $\mathbf{W}$, i.e., to calculate $\mathbf{W}(L)$. The length of the training sequence during initialization was set to $L = 300$, as in [2]. Setting the number of training epochs to 200 proved sufficient for the gradient descent algorithm, whereas five epochs were found sufficient for the ERLS learning algorithm. When the gradient descent algorithm was employed, $M = 5$ modules, each having $N = 2$ neurons, were used. This choice of parameters was chosen according to [2]. The ERLS algorithm was shown to perform best with $M = 3$ modules and $N = 1$ neurons. The number of the external inputs to a module of the PRNN using the GD learning algorithm was set to $p = 4$. Simulations show that increasing the system order does not improve the prediction performance. When the ERLS learning algorithm was used, however, the best results were achieved when parameter p was set to $p = 8$. The nonlinear activation function used by the neurons of the PRNN was the logistic sigmoid function given by

$$\Phi(v) = \frac{1}{1 + \exp(-bv)} \tag{35}$$

where the constant b controls the slope of the function. For the PRNN employing the gradient descent algorithm, the appropriate value for b was found to be $b = 1.0$, whereas for the PRNN using the ERLS learning algorithm, it was $b = 2.75$. The forgetting factors and the learning rates were chosen individually for each input signal. Table I gives insight into the differences in terms of the prediction gain R_p (34) between the Haykin–Li scheme [2] and the proposed modified nonlinear predictors. The simulation was undertaken using the input speech signal denoted by $s2$. Tables II–IV comprise the simulation results for all speech signals used in the experiments, namely, $s1$, $s2$, and $s3$. The prediction gains in Tables I–IV were denoted as follows.

- R_{PN} for the nonlinear predictor based on the PRNN configuration;
- R_{PRLS} for the RLS linear predictor;
- R_{PLMS} for the LMS linear predictor.

The following symbols are used in Table V for the parameter settings of the PRNN and the linear predictors.

- Ω_P forgetting factor for the ERLS algorithm in the PRNN;
- μ_P learning rate of the SG algorithm in the PRNN;
- Ω_L forgetting factor for the RLS algorithm in the linear subsection of the PRNN based nonlinear predictor;
- μ_L learning rate for the LMS algorithm in the linear subsection of the PRNN based nonlinear predictor;
- Ω forgetting factor of the linear structure RLS based predictor;
- μ learning rate of the linear structure LMS based predictor.

TABLE I
COMPARISON OF HAYKIN'S AND LI'S AND THE MODIFIED NONLINEAR PREDICTORS USING SIGNAL $s2$

non-linear predictor	R_{PN} original version	R_{PN} modified version
SG+LMS	9.49	9.49
SG+RLS	11.80	11.80
ERLS+RLS, N=2	13.08	13.40

A. Haykin–Li Nonlinear Predictor versus Proposed Nonlinear Predictor

This first experiment shows that the simplification in the learning algorithm of the nonlinear PRNN predictor, as proposed in Section III, does not result in a degradation of the prediction gain. In fact, a slight improvement in performance has been obtained after the modification, as shown in Table I, which might be seen as a result of better utilizing the newly introduced information in the updated feedback vector of the PRNN. The rows in Table I show a comparison between the Haykin–Li and modified version of the PRNN-based predictor in terms of the prediction gain R_{PN}. Although identical results were obtained for the GD-trained networks (first and second row in Table I), when the ERLS learning algorithm was used in the PRNN (third row in Table I), the modified version achieved an 0.32 dB advantage in the prediction gain R_{PN} over the version described in [2]. The rest of the experiments were therefore carried out using the modified and computationally more efficient version of the nonlinear predictor.

B. PRNN with Gradient Descent Learning Algorithm + LMS in the Linear Subsection

In this experiment, the nonlinear predictor using the gradient descent learning algorithm in the PRNN and the LMS algorithm in the linear section was examined. According to the results from the previous experiment, the prediction performance of the modified predictor was expected to be close to the performance of the Haykin–Li nonlinear predictor [2]. The results of the simulations for this experiment are shown in Tables II–IV. The same pattern for presenting information is used in the tables. The upper halves of the tables comprise the comparison of the performance of the nonlinear predictor using prediction algorithms to the performance of the sole linear LMS predictor, whereas in the lower halves of the tables, the comparison of the prediction algorithms against the sole RLS linear predictor is shown. The same notation as in previous tables was used, with N representing the number of neurons per module. The advantage of the PRNN + LMS nonlinear predictor over the conventional linear LMS predictor has been shown to be 1.01 dB for signal $s1$, 1.43 dB for signal $s2$, and 0.99 dB for signal $s3$ (see the sixth row in Tables II–IV).

TABLE II
COMPARISON OF NONLINEAR PREDICTOR WITH RLS AND LMS LINEAR PREDICTORS USING SIGNAL $s1$

non-linear predictor	R_{PN}	R_{PRLS}	$R_{PN} - R_{PRLS}$
SG+LMS	10.25	12.70	-2.45
SG+RLS	13.01	12.70	0.31
ERLS+RLS, N=1	14.73	12.70	2.03
ERLS+RLS, N=2	14.77	12.70	2.07
	R_{PN}	R_{PLMS}	$R_{PN} - R_{PLMS}$
SG+LMS	10.25	9.24	1.01
SG+RLS	13.01	9.24	3.77
ERLS+RLS, N=1	14.73	9.24	5.49
ERLS+RLS, N=2	14.77	9.24	5.53

TABLE III
COMPARISON OF NONLINEAR PREDICTOR WITH RLS AND LMS LINEAR PREDICTORS USING SIGNAL $s2$

non-linear predictor	R_{PN}	R_{PRLS}	$R_{PN} - R_{PRLS}$
SG+LMS	9.49	11.55	-2.06
SG+RLS	11.80	11.55	0.25
ERLS+RLS, N=1	13.59	11.55	2.04
ERLS+RLS, N=2	13.40	11.55	1.85
	R_{PN}	R_{PLMS}	$R_{PN} - R_{PLMS}$
SG+LMS	9.49	8.06	1.43
SG+RLS	11.80	8.06	3.74
ERLS+RLS, N=1	13.59	8.06	5.53
ERLS+RLS, N=2	13.40	8.06	5.34

Furthermore, the nonlinear predictor has been found to perform worse than the conventional linear RLS predictor for all of the speech signals being tested. The linear RLS predictor has shown an advantage of 2.45 dB over the nonlinear predictor for signal $s1$, 2.06 dB for signal $s2$, and 1.89 dB for signal $s3$ (see the first row in Tables II–IV). Moreover, the RLS is a computationally less complex algorithm than the gradient descent algorithm applied to the nonlinear predictor. This indicates that the nonlinear predictor is not sufficiently efficient in terms of computational requirements and prediction gain when the configuration with the GD learning algorithm in the PRNN and the LMS algorithm in the linear subsection is being used.

C. PRNN with Gradient Descent Learning Algorithm + RLS in the Linear Subsection

The previous experiment is repeated, with the only difference being that the RLS algorithm was used in the linear subsection instead of the LMS algorithm. The results of the simulations for the same three speech signals $s1$, $s2$, and $s3$ are shown in Tables II–IV following the same pattern as in the previous experiment. The prediction gains obtained were higher than the prediction gains achieved with the sole linear RLS predictor for all three signals. The advantage of the nonlinear predictor over the RLS linear predictor was 0.31 dB for signal $s1$, 0.25 dB for signal $s2$, and 0.05 dB for signal $s3$ (see the second row in Tables II–IV). Compared

TABLE IV
COMPARISON OF NONLINEAR PREDICTOR WITH RLS AND LMS LINEAR PREDICTORS USING SIGNAL $s3$

non-linear predictor	R_{PN}	R_{PRLS}	$R_{PN} - R_{PRLS}$
SG+LMS	7.3	9.19	-1.89
SG+RLS	9.24	9.19	0.05
ERLS+RLS, N=1	10.90	9.19	1.61
ERLS+RLS, N=2	9.85	9.19	1.11
	R_{PN}	R_{PLMS}	$R_{PN} - R_{PLMS}$
SG+LMS	7.30	6.31	0.99
SG+RLS	9.24	6.31	2.93
ERLS+RLS, N=1	10.80	6.31	4.49
ERLS+RLS, N=2	9.85	6.31	3.54

with the sole LMS linear predictor, the advantage of the nonlinear predictor for speech signals $s1$, $s2$, and $s3$ was, respectively, 3.77, 3.74, and 2.93 dB (see the seventh row in Tables II–IV). This shows that the advantage in the prediction gain achieved by using the nonlinear predictor compared with the sole RLS linear predictor is not substantial. At the same time, the computational complexity of the nonlinear predictor is much higher than the computational complexity of a linear RLS-based predictor.

D. PRNN with ERLS Learning Algorithm + RLS in the Linear Subsection

All the results presented so far, and the results obtained in [2], show little benefit using the complex PRNN architecture, as compared with the conventional RLS linear predictor. This was the authors' motivation to investigate possible alternatives in the learning algorithms. An algorithm that, when applied to the PRNN-based nonlinear predictor, results in considerable advantage in prediction gain over the sole RLS predictor is the ERLS learning algorithm. The results of the simulations with the ERLS learning algorithm in the nonlinear subsection and the RLS algorithm in the linear subsection of the nonlinear predictor are shown in Tables II–IV (see the fourth, fifth, eighth, and ninth rows). The highest prediction gains were achieved when the modules in the PRNN were working with only one neuron. In that case, the nonlinear predictor achieved approximately 2 dB for signals $s1$ and $s2$ and 1.61 dB for signal $s3$ advantage in prediction gain over the RLS linear predictor (see the third row in Tables II–IV).

For the signal $s1$, a slight increase in the prediction gain was noted when two neurons were used per module, whereas for signals $s2$ and $s3$, this resulted in performance deterioration (see the fourth and ninth rows in Tables II–IV). A reason for not benefiting in performance when the number of neurons per module is increased can be related to the features of the learning algorithm used. The ERLS algorithm is based on a linearization of the nonlinear mapping function in the observation equation. A first-order Taylor expansion [see (24)] of the nonlinear function $\mathbf{h}(\cdot)$ is used. If the first-order approximation to the function $\mathbf{h}(\cdot)$ is not sufficiently accurate, then the performance of the ERLS algorithm will not improve while increasing the number of neurons in a module. On

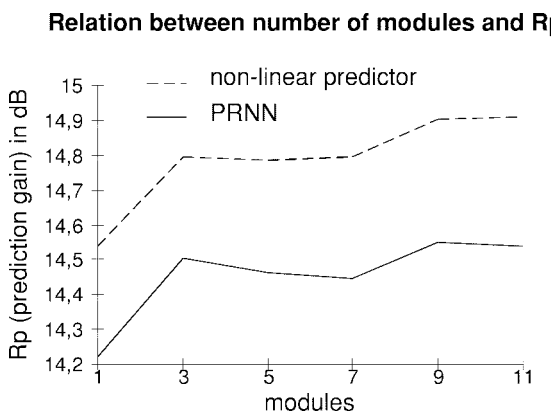

Fig. 4. Relation between the number of modules in PRNN and the prediction gain.

the other hand, increasing the number of neurons increases the inherent nonlinearity in the system. Therefore, the first-order Taylor expansion may not satisfy the demands of the nonlinear mapping function, which explains why an increase in the number of neurons per module does not necessarily lead to a better performance of the system.

This experiment shows that the nonlinear predictor achieves good prediction results when the ERLS algorithm was used in the PRNN, even when only one neuron was used in each module of the PRNN. On the other hand, except for the network dynamics, the computational complexity of the ERLS algorithm is of the same order of magnitude as that of the RTRL algorithm. Thus, the ERLS algorithm is indeed a more appropriate algorithm for training the PRNN for the prediction purpose.

E. The Contribution of the Linear Subsection Toward the Total Prediction Performance

The last experiment was undertaken again, whose outcome is shown in Fig. 4, which shows the relation between the number of modules used in the PRNN and the corresponding prediction gain for signal $s1$. The results are shown for both the sole PRNN and the whole nonlinear predictor, including the linear subsection. The contribution of the linear subsection to the total prediction gain can be found as the difference between the corresponding curves in Fig. 4. The difference was found to be approximately constant ($\approx$0.3 dB) for the numbers of modules considered. The same analysis for signals $s2$ and $s3$ has shown the difference in prediction gain of ($\approx$0.27 dB) for signal $s2$ and ($\approx$0.2 dB) for signal $s3$, which is the measure of the contribution of the linear subsection of the nonlinear predictor to the prediction gain of the whole nonlinear predictor. At the same time, the computational complexity of the linear subsection is not significant, as compared with the computational complexity of the PRNN. This supports the notion of first extracting nonlinearity from, e.g., speech signals and then processing them by conventional adaptive linear predictors.

Fig. 4 also shows that increasing the number of modules and, consequently, the degree of nonlinearity of the structure of the nonlinear predictor, does improve the performance of the predictor. The greatest improvements have been achieved when the number of modules was increased from one to three, thereby improving the prediction gain by 0.26 dB. A further increase in the number of modules would result in further improvements in the prediction gain. The advantage in prediction gain obtained, however, does not support the use of such a complex architecture (i.e., 0.11 dB is the advantage in the prediction gain for an increase in the number of modules of the PRNN from three to 11).

Finally, the PRNN-based nonlinear predictor was compared with the sole nonlinear predictor realized as a single recurrent neural network (RNN), consisting of the same number of neurons as the PRNN. The advantage in prediction gain with the structure of the nonlinear predictor with one neuron in three modules and the RLS algorithm in the linear subsection to a single RNN with the same number of neurons was 0.58 dB for signal $s1$, whereas for signals $s2$ and $s3$, the corresponding values were 0.53 and 0.98 dB, respectively.

VI. Conclusion

New learning algorithms for an adaptive nonlinear forward predictor based on a pipelined recurrent neural network (PRNN) have been presented. A gradient descent (GD) learning algorithm, which has lower computational complexity then a standard real time recurrent learning (RTRL) algorithm for the PRNN architecture, has been developed. This GD algorithm, when applied to the PRNN structure, has been demonstrated not to outperform, in terms of prediction gain, a conventional adaptive forward predictor based on a recursive least squares (RLS) algorithm.

An extended recursive least squares (ERLS) training algorithm for the PRNN has therefore been developed. For a class of highly nonstationary time-varying signals, such as speech, the ERLS training algorithm has been shown to obtain significantly higher prediction gain values than the previously developed GD algorithm [2], provided that the underlying linearization of the PRNN structure was appropriate. With the ERLS learning algorithm, the nonlinear predictor achieved advantages of approximately 2 dB in terms of prediction gain over the sole linear RLS predictor. This algorithm performed best with only one neuron in each of the three PRNN modules and is thus computationally less demanding than the nonlinear predictor presented in [2], which used two neurons and five modules. The benefit in the total number of neurons was due to the strength of the novel ERLS algorithm.

Compared with the recurrent neural network (RNN) with the same number of neurons as the PRNN, the PRNN architecture exhibits better prediction gains with less computational complexity.

The above results demonstrate the potential of the combined linear and nonlinear processing architectures in prediction applications.

Acknowledgment

The authors would like to thank Dr. Li for providing the speech signal $s2$, which was used in the simulations, and the anonymous reviewers for their helpful comments.

REFERENCES

[1] J. Makhoul, "Linear prediction: A tutorial review," *Proc. IEEE*, vol. 63, pp. 561–580, Apr. 1975.

[2] S. Haykin and L. Li, "Non-linear adaptive prediction of nonstationary signals," *IEEE Trans. Signal Processing*, vol. 43, pp. 526–535, Feb. 1995.

[3] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[4] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[5] R. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, pp. 270–280, 1989.

[6] J. Shynk, "Adaptive IIR filtering," *IEEE Acoust., Speech, Signal Processing Mag.*, vol. 6, pp. 4–21, Apr. 1989.

[7] http://www.ert.rwth-aachen.de/Personen/balterse.html.

Jens Baltersee was born in Essen, Germany, on April 25, 1971. He received the M.Eng. degree in electrical and electronic engineering from Imperial College of Science, Technology, and Medicine, London, U.K., in 1996.

Currently, he is a Research Assistant at the Institute for Integrated Signal Processing Systems (ISS), Aachen University of Technology, Aachen, Germany, working toward the Ph.D. degree. His current research interests include synchronization, estimation, equalization, and space-time processing.

Jonathon A. Chambers (M'93) was born in Peterborough, U.K., in 1960. After an electronics artificer apprenticeship in the Royal Navy, he received the first class B.Sc. (Hons.) degree in electrical and electronic engineering from the Polytechnic of Central London, London, U.K., receiving the Robert Mitchell Medal as the top graduate in 1985. He was then appointed to a lectureship at the Polytechnic of Central London, teaching courses in digital signal processing. He received the Ph.D. degree in adaptive signal processing in 1990 from Imperial College, London, U.K. after studying there and at Cambridge University, Cambridge, U.K.

He spent three years as a Research Scientist at Schlumberger Cambridge Research, applying adaptive signal processing techniques to oilfield-related applications. He returned to a lectureship in signal processing in the Department of Electrical and Electronic Engineering, Imperial College, in 1994. He has authored or coauthored many technical publications on adaptive signal processing and is currently running research projects in adaptive signal processing and its application in mobile communication systems.

Dr. Chambers is a member of the IEE Professional Group Committee E5 on Signal Processing, a Guest Editor for the *International Journal of Adaptive Control and Signal Processing*, and an Associate Editor for IEEE TRANSACTIONS ON SIGNAL PROCESSING.