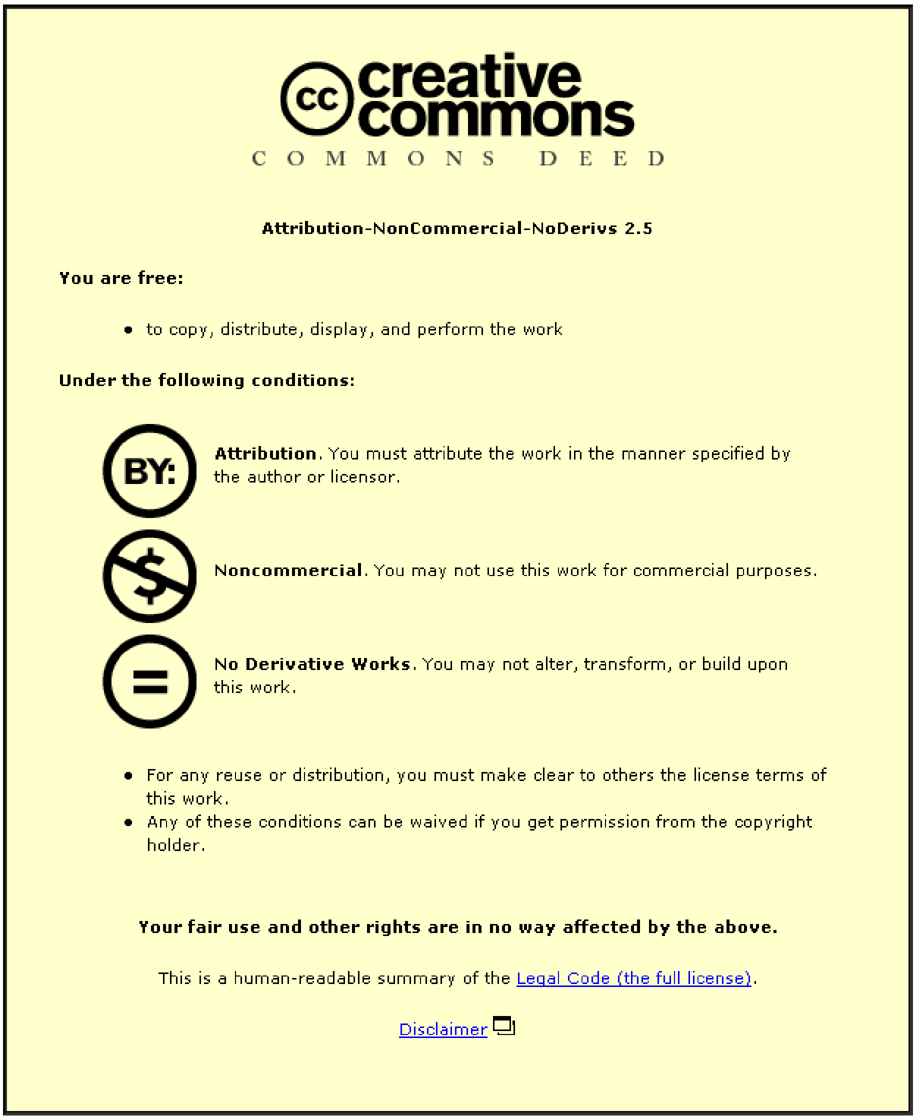


From An *A Priori* RNN to An *A Posteriori* PRNN Nonlinear Predictor

Danilo P. Mandic and Jonathon A. Chambers

Signal Processing and Digital Systems Section
Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine
Exhibition Road, SW7 2BT London
E–mail: d.mandic@ic.ac.uk, j.chambers@ic.ac.uk

Abstract

We provide an analysis of nonlinear time series prediction schemes, from a common Recurrent Neural Network (RNN) through to the Pipelined Recurrent Neural Network (PRNN), which consists of a number of nested small–scale RNNs. All these schemes are shown to be suitable for Nonlinear Autoregressive Moving Average (NARMA) prediction. The time management policy of such prediction schemes is addressed and classified in terms of *a priori* and *a posteriori* mode of operation. Moreover, it is shown that the basic *a priori* PRNN structure exhibits certain *a posteriori* features. In search for an optimal PRNN based predictor, some inherent features of the PRNN, such as nesting and the choice of cost function are addressed. It is shown that nesting in essence is an *a posteriori* technique which does not diverge. Simulations undertaken on a speech signal support the algorithms derived, and outperform linear Least Mean Square (LMS) and Recursive Least Squared (RLS) predictors.

1 Introduction

An important area in signal processing is time series prediction, which has traditionally been achieved through linear structures, such as parametric Autoregressive (AR), Moving Average (MA), or Autoregressive Moving Average (ARMA) models [1, 2]. Among linear adaptive time series predictors, the most important examples are Least Mean Square (LMS) and Recursive Least Squares (RLS) predictors [3]. For stochastic signals with nonstationary statistical characteristics that cannot be adequately processed by linear predictors, a number of nonlinear predictors has been developed [4, 5]. An important class of nonlinear predictors is Artificial Neural Network (ANN) based nonlinear predictors [6, 7]; in particular Recurrent Neural Network (RNN) based predictors have been shown to be able to represent the Nonlinear Autoregressive Moving Average (NARMA) process [8]. RNN parameters are commonly adapted by the use of the Real Time Recurrent Learning (RTRL) algorithm [9], whose computational complexity is $\mathcal{O}(N^4)$, where N is the number of neurons in the RNN, which is rather demanding. Therefore, if a prediction scheme can be found which would outperform a common RNN prediction scheme, without additional computational complexity, it would be of great

0-7803-5060-X/98/$10.00 © 1998 IEEE

benefit. Here, we present two such approaches, namely *a posteriori* predictors based upon the RNN, and nested schemes, whose representative is the Pipelined Recurrent Neural Network (PRNN) [10]. Further, it is shown that these two schemes under certain conditions considerably overlap. Both of the schemes have tolerable computational complexity, which for the *a posteriori* approach, involves only an additional multiply–sum of order $\mathcal{O}(1)$, whereas for the PRNN, which is a nested structure of a number of small–scale RNNs computational complexity is $\mathcal{O}\left((M \times N)^4\right)$.

2 NARMA Processes and Recurrent Neural Networks

According to [5], a nonlinear system can be defined by a NARMA difference equation

$$x(t) = e(t) + h\left(x(t-1), \cdots, x(t-p), e(t-1), \cdots, e(t-q)\right) \qquad (1)$$

where p denotes the order of the Autoregressive (AR) part, and q denotes the order of the Moving Average (MA) part. A number of stochastic signal models have been developed by appropriately defining the nonlinear function $h(\cdot)$ so as the estimate $\hat{x}(t) = E\left(x(t)\right)$ exhibits certain behaviour. Since the innovation process $\{e(t)\}$ is not observable, the residual $\hat{e}(t) = x(t) - \hat{x}(t)$, is an emergent approximation which can be used instead of $e(t)$ in (1). If, in order to match the notation common in RNNs, we denote the predicted values $\hat{x}$ by y, and have $y(n) = \hat{x}(n)$, the NARMA scheme from (1), can be further approximated as [6]

$$\begin{aligned} y(n) &= h(x(n-1), \cdots, x(n-p), \hat{e}(n-1), \cdots, \hat{e}(n-q)) \\ &= h(x(n-1), \cdots, x(n-p), (x(n-1) - y(n-1)), \cdots \\ &\quad \cdots, (x(n-q) - y(n-q))) \\ &= H(x(n-1), \cdots, x(n-p), y(n-1), \cdots, y(n-q)) \qquad (2) \end{aligned}$$

where H is some new, nonlinear smooth function. The last equation in (2) is now suitable for the RNN implementation, with H becoming an activation function of the neuron, which is typically the logistic function denoted by

$$\Phi(v) = \frac{1}{1 + e^{-bv}} \qquad (3)$$

and will be assumed in (4). The realisation of the process (2) by an RNN is shown in Figure 1. The set of equations which fully describe the RNN given in Figure 1 is

$$\begin{aligned} y_i(k) &= \Phi\left(v_i(k)\right), \quad i = 1, \cdots, N \\ v_i(k) &= \sum_{l=1}^{p+q+N+1} w_{i,l}(k) x_l(k), \quad i = 1, \cdots, N \end{aligned}$$

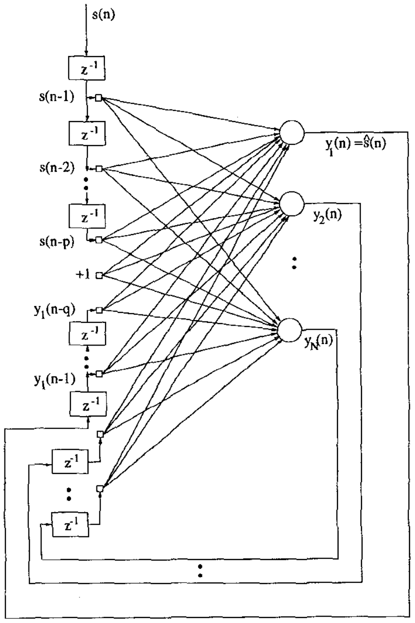


Figure 1: Recurrent NARMA(p,q) implementation of prediction model

$$\begin{aligned} x_l(k) \in \mathbf{X}^t(k) &= [y_1(k-1), \cdots, y_1(k-q), \\ & \quad y_2(k-1), \cdots, y_N(k-1), 1, s(k-1), \cdots, s(k-p)] \quad (4) \end{aligned}$$

where the unity element corresponds to the bias input to the neurons. The output values of neurons in the RNN are denoted by $y_1, \cdots, y_N$ and external input signal samples by s. The set of weights $\{w_{i,l}(k)\}$ for every neuron y_i, $i = 1, \cdots, N$ are arranged in the weight matrix $\mathbf{W}(k)$.

2.1 Training Process for the NARMA(p,q) Predictor

RTRL based training of the RNN is based upon minimising the instantaneous squared error at the output of the first neuron of the RNN [9], which can be expressed as $min(e^2(n)) = min([s(n) - y_1(n)]^2)$, where $e(n)$ denotes the error at the output of the RNN, and $s(n)$ is the teaching signal. The correction $\Delta\mathbf{W}(n)$ to the weight matrix $\mathbf{W}(n)$ of the RNN is calculated as

$$\Delta\mathbf{W}(n) = -\eta \frac{\partial e^2(n)}{\partial \mathbf{W}(n)} = 2\eta e(n) \frac{\partial y_1(n)}{\partial \mathbf{W}(n)} \quad (5)$$

which turns out to be a recursive calculation of the gradients of the outputs of the neurons [9, 11]. In order to make the algorithm run in real-time, an approximation has to be made, namely that for a small learning rate η, the following approximation

$$\frac{\partial y_1(n-1)}{\partial \mathbf{W}(n)} \approx \frac{\partial y_1(n-1)}{\partial \mathbf{W}(n-1)} \tag{6}$$

holds for slowly varying systems. However, in a class of RNN networks, where a number of delayed output signals is fed back into the RNN (Figure 1), the approximation which has to be made is

$$\frac{\partial y_1(n-i)}{\partial \mathbf{W}(n)} \approx \frac{\partial y_1(n-i)}{\partial \mathbf{W}(n-i)}\ , \quad i = 1, 2, \cdots, q \tag{7}$$

which might not be appropriate, even for small learning rate η. Hence, having in mind the importance of the signal prediction paradigm, there is a need for another learning strategy, which would possibly overcome those difficulties encountered in traditional RTRL learning. One solution would be to increase the number of neurons N in the RNN, but that would involve considerably increased computational complexity, which is $\mathcal{O}(N^4)$ for the RNN [9]. A solution to the problem stated might be the *a posteriori* approach, which utilises the issue of time management policy throughout the RTRL algorithm in order to improve the prediction. Another, solution would be the PRNN approach, whose computational complexity for M modules with N neurons is $\mathcal{O}(M \times N^4) << \mathcal{O}\left((MN)^4\right)$.

2.2 The *A Posteriori* Approach for the RNN Based Prediction

The output of the RNN, denoted by y, can be expressed as

$$y(k) = \Phi\left(\mathbf{X}^t(k)\mathbf{W}(k)\right) \tag{8}$$

where the input vector to the RNN can be expressed as $\mathbf{X}(k) = [\{y\}, 1, \{s\}]^t$, where s denotes the external input signal to be predicted. As the updated weight matrix $\mathbf{W}(k+1)$ is available before the arrival of the next, i.e. *updated*, input $\mathbf{X}(k+1)$ to the RNN, an improved, *a posteriori* estimate $\bar{y}$ can be formed as

$$\bar{y}(k) = \Phi\left(\mathbf{X}^t(k)\mathbf{W}(k+1)\right) \tag{9}$$

Note that $\bar{y}(k)$ in (9) is, strictly speaking, no longer causal, since the elements of the weight matrix $\mathbf{W}(k+1)$ were used to calculate $\bar{y}(k)$. The *a posteriori* output value $\bar{y}$ was obtained in an *iterative* fashion, i.e. applying the existing input vector $\mathbf{X}(k)$ to the RNN based on the newly calculated weight matrix $\mathbf{W}(k+1)$, which is *recursively* calculated through the RTRL algorithm. Thus, using a combination of *recursive* and *iterative* signal processing, an improved prediction scheme can be applied, without substantially increased computational complexity. If the desirable scheme (9) could be made strictly causal, and consequently realisable, the *a posteriori* approach would be preferable as compared to the *a priori* approach. It can be shown that the *a posteriori* prediction error $\bar{e}(k) = \bar{y}(k) - s(k)$, where $\{\bar{y}\}$ and $\{s\}$ now build $\bar{\mathbf{X}}$, can be expressed as

$$\bar{e}(k) \approx \frac{d(k) - \Phi\left(\bar{\mathbf{X}}^t(k)\mathbf{W}(k)\right)}{1 + 2\eta\dot{\Phi}\bar{\mathbf{X}}^t(k)\mathbf{\Pi}(k)} \tag{10}$$

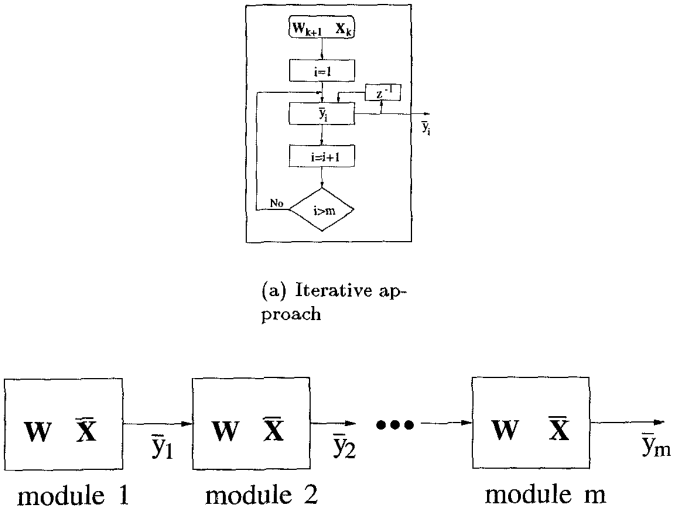
The *a posteriori* prediction error $\bar{e}(k)$ in (10) does not comprise the terms with the index $(k+1)$, which makes the learning process causal. The dominator in (10) is strictly greater than unity, and serves as a stabilising term, providing the Nonlinear Normalised LMS (NNLMS) features, and may become relatively big in magnitude, since the gradients of the neurons $\mathbf{\Pi}$ can be high. Hence, using an *a posteriori* recurrent nonlinear module in the PRNN, it is possible to obtain lower prediction error, than using an *a priori* recurrent module. Iteration (9) can be repeated, and can be shown not to diverge [12]. However, there might not be enough time between two consecutive input samples for a sufficient number, say M, iterations. In other words, if τ is the time needed for one iteration to complete, the aim is to enable the iterative process (9) to finish in time measure of $\mathcal{O}(\tau)$, rather than $\mathcal{O}(M \times \tau)$. Such a strategy is known as *pipelining* and is widely used in advanced computer architectures. Using the *pipelining* strategy, a task is divided in subtasks, each of them is represented by a module. The modules exchange their output values and most often are merely cascaded. The concept of iterative and pipelining approach is shown in Figure 2.

3 The Pipelined Recurrent Neural Network (PRNN)

The PRNN is a modular neural network, and consists of a certain number M of RNNs as its modules, with each module consisting of N neurons. In the PRNN configuration, the M modules, which are RNNs, are connected as shown in Figure 3. The $(p \times 1)$ dimensional external signal vector $\mathrm{s}^T(n) = [s(n-1), \cdots, s(n-p)]$ is delayed by m time steps $(z^{-m}\mathbf{I})$ before feeding the module m. All the modules operate using the same weight matrix $\mathbf{W}$. The overall output signal of the PRNN is $y_{out}(n) = y_{1,1}(n)$, i.e. the output of the first neuron of the first module. Thus, the overall cost function of the PRNN becomes

$$E(n) = \sum_{i=1}^{M} \lambda^{i-1} e_i^2(n) \tag{11}$$

where $e_i(n)$ is the error from module i, and a *forgetting factor* λ, $\lambda \in (0,1]$, is introduced which determines the weighting of the individual modules.



(a) Iterative approach

(b) Pipelined approach

Figure 2: From iterative to *pipelined* fixed point iteration

3.1 An Analysis of the Influence of the Forgetting Factor on the Total Prediction Gain

According to the method of recursive least squares, the error criterion which requires minimisation is

$$E(t) = \sum_{\tau=0}^{t} \beta^{\tau} \hat{e}^2(t-\tau) \longrightarrow minimum \tag{12}$$

with weighting factor $0 < \beta \leq 1$. The calculation of $E(t)$ can be carried out recursively as

$$E(t) = \beta E(t-1) + \hat{e}^2(t) \tag{13}$$

However, as the processes $\{y_1(n)\}, \{y_2(n)\}, \cdots, \{y_M(n)\}$ at the output of the modules of the PRNN do not represent realisations of the same stochastic process, the cost function (11) might not be the best choice. Indeed, the updating of the weight matrix $\mathbf{W}$ of the PRNN can be expressed as

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \Delta\mathbf{W}(n) = \mathbf{W}(n) + \sum_{i=1}^{M} \lambda^{i-1} \Delta\mathbf{W}_i(n) \tag{14}$$

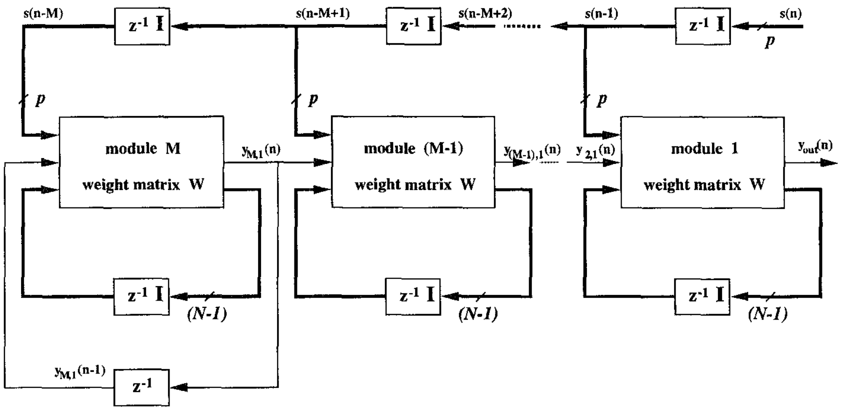


Figure 3: Pipelined recurrent neural network

where the elements of the correction $\Delta\mathbf{W}$ to the weight matrix $\mathbf{W}$ can be calculated as

$$\Delta w_{k,l}(n) = -\eta \frac{\partial}{\partial w_{k,l}(n)} \left(\sum_{i=1}^{M} \lambda^{i-1} e_i^2(n) \right) = -2\eta \sum_{i=1}^{M} \lambda^{i-1} e_i(n) \frac{\partial e_i(n)}{\partial w_{k,l}(n)} \tag{15}$$

Function (11) provides MA smoothing of its arguments (since $\lambda^i > 0, i = 1, \cdots, M, \lambda > 0$). Hence, it is the forgetting factor λ that has an influence on the learning process.

3.2 Nesting Process in the PRNN

A simple nesting scheme for functions with one variable is given by

$$\hat{x} = \Phi(x_m) = \Phi(\Phi(x_{m-1})) = \cdots = \Phi(\Phi(\Phi(\cdots(\Phi(x_1))\cdots) \tag{16}$$

and can be shown under some mild conditions not to diverge. Notice that the nesting process (16) represents an implicitly written iterative process, which is correspondent to the *a posteriori* approach

$$x_{i+1} = \Phi(x_i) \Leftrightarrow x_{i+1} = \Phi(\Phi(x_{i-1})) = \Phi(\Phi(\Phi(\cdots(\Phi(x_1))\cdots) \tag{17}$$

Let us therefore just show the diagram of the effects of the nesting process for the logistic function with slope $b = 1$, depicted in Figure 4. From Figure 4, it is apparent that nesting (16) provides influence on the amplitude of its argument. Hence, it is expected that the nesting process (16) with m stages on $[a, b]$ converges towards the interval $[|\Phi'(x^*)|^m a, |\Phi'(x^*)|^m b]$ [12], which provides the *a posteriori* features spatially, rather than temporally. It is now straightforward, that a PRNN description based upon (4) exhibits nesting.

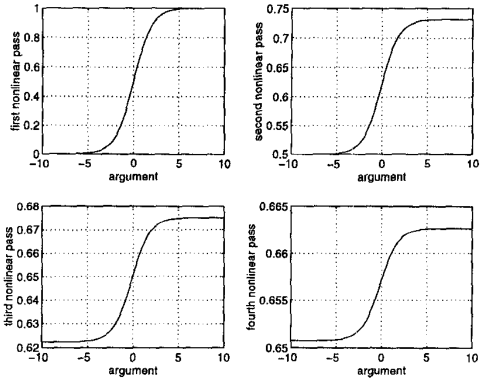


Figure 4: Nested *logistic* nonlinearity

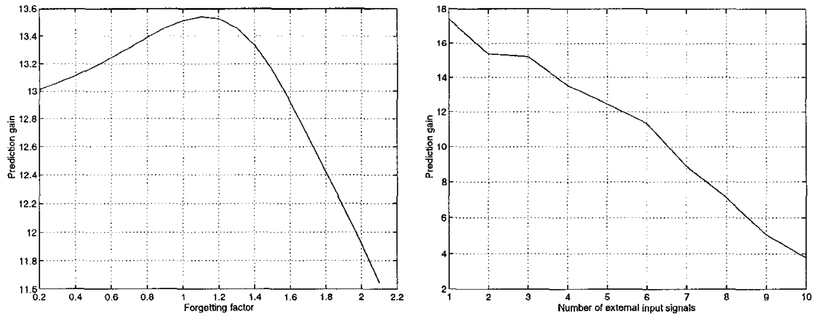
4 Experimental Results

In order to support the algorithms derived, simulations were undertaken on a speech signal denoted by $s1$. The measure that was used to assess the performance of the predictors was the forward prediction gain R_p given by

$$R_p \triangleq 10\,log_{10}\left(\frac{\hat{\sigma}_s^2}{\hat{\sigma}_e^2}\right) dB \qquad (18)$$

where $\hat{\sigma}_s^2$ denotes the estimated variance of the speech signal $\{s(n)\}$, whereas $\hat{\sigma}_e^2$ denotes the estimated variance of the forward prediction error signal $\{e(n)\}$.

Simulations undertaken on $s1$ include LMS prediction, RLS prediction, *a priori* and *a posteriori* RNN prediction, and *a priori* and *a posteriori* PRNN prediction. The results of simulations are shown in a self-explanatory way in Table 1. The relationship between the prediction gain Rp and the value of λ in (11) for signal $s1$, having the PRNN with $p = 4, M = 5, \eta = 0.07, N = 2$ is given in Figure 5(a). From Figure 5(a), the best value for the smoothing factor λ for signal $s1$ is $\lambda_{opt} = 1.1$, where prediction gain $Rp = 13.54dB$. In Figure 5(b), relationship between prediction gain Rp versus the number of input signals to the PRNN is shown. If instead of $p = 4$, we put $p = 1$, the results achieved are even better than with $p = 4$. Now, it is possible to apply the *a posteriori* approach on the *a priori* PRNN structure, which again takes the benefits of the time management policy and exhibits improved results.



(a) Prediction gain Rp versus the forgetting factor λ for $s1$

(b) Prediction gain Rp versus the number of external inputs p for speech signal $s1$

Figure 5: Towards optimal PRNN parameters for $s1$

Table 1: Prediction gains for the predictors addressed

speech signal	$s1$
$Rp[dB]$ LMS only	8.06
$Rp[dB]$ RLS only	11.55
$Rp[dB]$ *a priori* RNN	8.48
$Rp[dB]$ *a posteriori* RNN	8.79
$Rp[dB]$ for *a priori* PRNN, $p = 4$, $\lambda = 1.1$	12.48
$Rp[dB]$ for *a posteriori* PRNN, $p = 4$, $\lambda = 1.1$	13.54
$Rp[dB]$ for *a posteriori* PRNN, $p = 1$, $\lambda = 1.1$	17.66

5 Conclusions

Temporal and spatial methods to improve existing RNN predictors have been addressed. It is shown that for an RNN architecture, the *a posteriori* approach gives better results, as compared to the *a priori* one. On the other hand, nesting realised through the Pipelined Recurrent Neural Network (PRNN) offers a spatial representation of the same problem. The *a posteriori* approach does not involve additional order of computational complexity, while the PRNN approach, whose computational complexity is $\mathcal{O}(M \times N^4)$ offers improved prediction performance, whilst maintaining low computational complexity as compared to the RNN ($\mathcal{O}\left((MN)^4\right)$). The results of simulations support the algorithms presented.

References

[1] J. Makhoul, "Linear prediction: A tutorial overview," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 561–580, 1975.

[2] G. E. Box and G. M. Jenkins, *Time series analysis: forecasting and control.* Holden–Day, second ed., 1976.

[3] J. R. Treichler, C. R. Johnson, Jr., and M. G. Larimore, *Theory and Design of Adaptive Filters.* John Wiley & Sons, 1987.

[4] V. Mathews, "Adaptive polynomial filters," *IEEE Signal Processing Magazine*, vol. 8, no. 3, pp. 10–26, 1991.

[5] M. Priestley, *Non–linear and Non–stationary Time Series Analysis.* Academic Press, London, 1991.

[6] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, 1994.

[7] R. M. Dillon and C. N. Manikopoulos, "Neural net nonlinear prediction for speech data," *Electronics Letters*, vol. 27, no. 10, pp. 824–826, 1991.

[8] L. Li, "Approximation theory and recurrent networks," in *Proceedings of the International Joint Conference on Neural Networks*, vol. II, pp. 266–271, 1992.

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

[10] S. Haykin and L. Li, "Nonlinear adaptive prediction of nonstationary signals," *IEEE Transactions on Signal Processing*, vol. 43, no. 2, pp. 526–535, 1995.

[11] A. Prochazka and P. Sovka, eds., *Signal Analysis and Prediction*, ch. D. P. Mandic, J. Baltersee, and J. A. Chambers: Non–linear Adaptive Prediction of Speech with a Pipelined Recurrent Neural Network and Advanced Learning Algorithms. Birkhauser, Boston, 1998.

[12] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice–Hall Series in Computational Mathematics, 1983.