PERGAMON

AVAILABLE AT
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT•

Neural Networks 16 (2003) 1095–1100

Neural Networks

www.elsevier.com/locate/neunet

Neural Networks letter

Recurrent neural networks with trainable amplitude of activation functions

Su Lee Goh*, Danilo P. Mandic

Imperial College of Science, Technology and Medicine, London SW7 2AZ, UK

Accepted 9 May 2003

Abstract

An adaptive amplitude real time recurrent learning (AARTRL) algorithm for fully connected recurrent neural networks (RNNs) employed as nonlinear adaptive filters is proposed. Such an algorithm is beneficial when dealing with signals that have rich and unknown dynamical characteristics. Following the approach from [Trentin, E. *Network with trainable amplitude of activation functions,* Neural Networks 14 (2001) 471], three different cases for the algorithm are considered; a common adaptive amplitude shared among all the neurons; each layer has its own adaptive amplitude; different adaptive amplitude for each neuron. Experimental results show the AARTRL outperforms the standard RTRL algorithm.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Nonlinear adaptive filters; Recurrent neural networks; Adaptive amplitude

1. Introduction

The most frequently applied neural network architectures are the feedforward networks and recurrent networks. Feedforward (FF) networks represent static nonlinear models and can have multilayer architecture. Recurrent networks are dynamic networks and their structures are fundamentally different from the static ones, because they contain feedbacks (Mandic & Chambers, 2001). Recurrent neural networks (RNNs) can yield smaller structures than nonrecursive feedforward neural networks in the same way that infinite impulse response (IIR) filters can replace longer finite impulse response (FIR) filters. The feedback characteristics of RNNs enable them to acquire state representation, which make them suitable for applications such as nonlinear prediction and modelling, adaptive equalization of communication channels, speech processing, plant control and automobile engine diagnostics (Haykin, 1999). Thus, RNNs offer an alternative to the dynamically driven feedforward networks (Haykin, 1999). RNNs have traditionally been trained by the Real Time Recurrent Learning (RTRL) algorithm (Williams & Zipser, 1989) which provides the training process of the RNN. Another algorithm referred to as Back-propagation-through-time (BPTT) can also be used for training the RNN. However, due to the computational complexity of the BPTT algorithm compared to the more simpler and efficient RTRL algorithm, we will be using the RTRL algorithm to train the RNN. In the paper by Trentin (2001), he has showed the importance of having adaptive amplitude of activation functions for the case of FF neural networks. Here, we embark upon the concept by Trentin and extend the derivation of the RTRL algorithm and introduce trainable amplitude of the activation function. This is important for nonlinear activation function since this way there is no need to standardised and rescale input signals to match the dynamical range of the activation function. The RTRL equipped with the adaptive amplitude can be employed where the dynamical range of the input signal is not known a priori. This way, the Adaptive Amplitude Real Time Recurrent Learning (AARTRL) algorithm is derived. The analysis is supported by examples of nonlinear prediction and Monte Carlo Analysis in which the AARTRL outperforms the RTRL.

2. Derivation of the fully connected recurrent neural network for the formulation of the real time learning algorithm

The set of equations which describe the RNN given in Fig. 1 is

$$y_i(k) = \Phi(v_i(k)), \qquad i = 1, \ldots, N \tag{1}$$

* Corresponding author.
E-mail addresses: su.goh@imperial.ac.uk (S.L. Goh), d.mandic@imperial.ac.uk (D.P. Mandic).

0893-6080/03/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0893-6080(03)00139-4

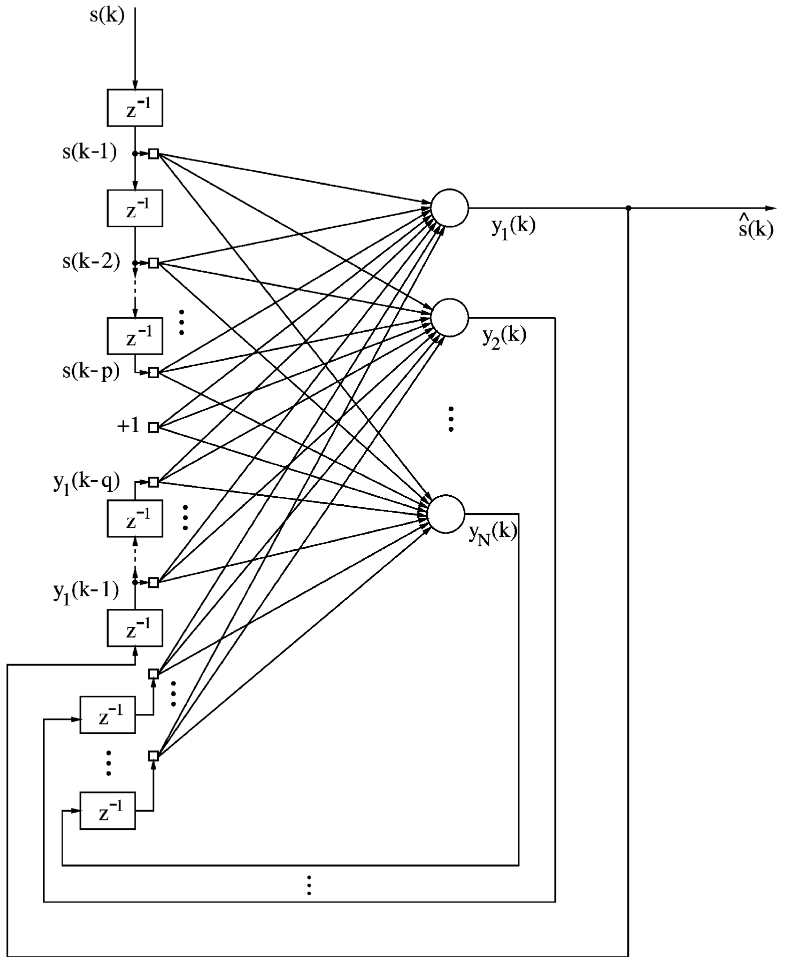

Fig. 1. Full recurrent neural network for prediction.

$$v_i(k) = \sum_{n=1}^{p+q+N} w_{i,n}(k)u_n(k) \tag{2}$$

$$\begin{aligned}\mathbf{u}_i^{\mathrm{T}}(k) =&[s(k-1),\ldots,s(k-p),1,y_1(k-1),\ldots,y_1\\ &(k-q),y_2(k-1),\ldots,y_N(k-1)]\end{aligned} \tag{3}$$

where the unity element in Eq. (3) corresponds to the bias input to the neurons. The output values of the neurons are denoted by $y_1,\ldots,y_N$ and the input samples are given by s. For the nth neuron, its weights form a $(p+q+N)\times 1$ dimentional weight vector $\mathbf{w}_i^{\mathrm{T}} = [w_{i,1},\ldots,w_{i,p+q+N}]$, where p is the number of external inputs, q the feedback connection of the first neuron and N is the number of neurons in the RNN. RTRL based training of the RNN for nonlinear adaptive filtering is based upon minimising the instantaneous squared error at the output of the first neuron of the RNN (Williams & Zipser, 1989), which can be expressed as $\min(e^2(k)) = \min([s(k)-y_1(k)]^2)$, where $e(k)$ denotes the error at the output of the RNN and $s(k)$ is the desired signal. This way, for a cost function $E(k) = \frac{1}{2}e^2(k)$, the weight matrix update becomes

$$\Delta w_{i,n}(k) = -\eta\frac{\partial E(k)}{\partial w_{i,n}(k)} = -\eta e(k)\frac{\partial e(k)}{\partial w_{i,n}(k)} \tag{4}$$

Since the external signal vector **s** does not depend on the elements of **w**, the error gradient becomes

$$\frac{\partial e(k)}{\partial w_{i,n}(k)} = -\frac{\partial y_1(k)}{\partial w_{i,n}(k)} \tag{5}$$

This can be rewritten as

$$\begin{aligned}\frac{\partial y_1(k)}{\partial w_{i,n}(k)} &= \Phi'(v_1(k))\frac{\partial v_1(k)}{\partial w_{i,n}(k)}\\ &= \Phi'(v_1(k))\left(\sum_{i=1}^{N}\frac{\partial y_i(k-1)}{\partial w_{i,n}(k)}w_{1,i+p+q}(k)+\delta_{in}u_n(k)\right)\end{aligned} \tag{6}$$

where

$$\delta_{\mathrm{in}} = \begin{cases}1 & i=n\\ 0 & i\neq n\end{cases} \tag{7}$$

Under the assumption, also used in the RTRL algorithm (Williams & Zipser, 1989), that when the learning rate η is sufficiently small, we have

$$\frac{\partial y_i(k-1)}{\partial w_{i,n}(k)} \approx \frac{\partial y_i(k-1)}{\partial w_{i,n}(k-1)}, \ i=1,\ldots,N \quad (8)$$

$$\frac{\partial y_1(k-i)}{\partial w_{i,n}(k)} \approx \frac{\partial y_1(k-i)}{\partial w_{i,n}(k-i)}, \ i=1,2,\ldots,q \quad (9)$$

Introducing a triply indexed set of variables $\pi^j_{i,n}$ to characterize the RTRL algorithm for the RNN, the sensitivities

$$\frac{\partial y_j(k)}{\partial w_{i,l}(k)}$$

(Haykin, 1999; Williams & Zipser, 1989) become

$$\pi^j_{i,n}(k) = \frac{\partial y_j(k)}{\partial w_{i,n}(k)} \ 1 \le j,i \le N, 1 \le n \le p+q+N \quad (10)$$

$$\pi^j_{i,n}(k+1) = \Phi'(v_j(k)) \left[\sum_{m=1}^{N} w_{j,m}(k)\pi^m_{i,n}(k) + \delta_{ij}u_n(k) \right] \quad (11)$$

with initial conditions

$$\pi^j_{i,n}(0) = 0 \quad (12)$$

To simplify the presentation, we introduce three new matrices, the $N \times (N+p+q)$ matrix $\mathbf{\Pi}_i(k)$, the $N \times (N+p+q)$ matrix $\mathbf{U}_j(k)$, and the $N \times N$ diagonal matrix $\mathbf{F}(k)$ as (Haykin, 1999)

$$\mathbf{\Pi}_j(k) = \frac{\partial \mathbf{y}(k)}{\partial \mathbf{w}_j(k)}, \ \mathbf{y} = [y_1(k),\ldots,y_N(k)], j=1,2,\ldots,N \quad (13)$$

$$\mathbf{U}_j(k) = \begin{bmatrix} 0 \\ \vdots \\ \mathbf{u}(k) \\ \vdots \\ 0 \end{bmatrix} \leftarrow jth\ row,\ j = 1,\ldots,N \quad (14)$$

$$\mathbf{F}(k) = \mathrm{diag}(\Phi'(\mathbf{u}^{\mathrm{T}}(k)\mathbf{w}_1(k)),\ldots,\Phi'(\mathbf{u}^{\mathrm{T}}(k)\mathbf{w}_N(k))) \quad (15)$$

$$\mathbf{\Pi}_j(k+1) = \mathbf{F}(k)[\mathbf{U}_j(k) + \mathbf{W}_a(k)\mathbf{\Pi}_j(k)] \quad (16)$$

where $\mathbf{W}_a$ denotes the set of those enteries in $\mathbf{W}$ which correspond to the feedback connections.

3. Derivation of gradient-based algorithms to learn amplitudes

The AARTRL algorithm relies on the amplitude of the nonlinear activation function to be adaptive according to the change in the dynamics of the input signals. Extending the approach from Trentin (2001), we can express the activation function as

$$\Phi(k) = \lambda_i(k)\bar{\Phi}(k) \quad (17)$$

where λ_i denotes the amplitude of the nonlinearity, $\Phi(k)$, whereas $\bar{\Phi}(k)$ denotes the activation function with a unit amplitude. Thus if $\lambda_i = 1$ it follows that $\Phi(k) = \bar{\Phi}(k)$. The update for the gradient adaptive amplitude is given by (Trentin, 2001)

$$\lambda_i(k+1) = \lambda_i(k) - \rho \nabla_{\lambda_i(k)} E_i(k) \quad (18)$$

$$E_i(k) = \tfrac{1}{2}e_i^2(k) = \tfrac{1}{2}[s(k) - y_i(k)]^2 \quad (19)$$

where $\nabla_{\lambda_i(k)} E_i(k)$ denotes the gradient of the cost function with respect to the amplitude of the activation function λ and ρ denotes the step size of the algorithm and is a small constant. From Eq. (19), the gradient can be obtained as

$$\nabla_{\lambda_i(k)} E_i(k) = \frac{\partial E_i(k)}{\partial \lambda_i(k)} = e_i(k)\frac{\partial e_i(k)}{\partial \lambda_i(k)} = -e_i(k)\frac{\partial y_i(k)}{\partial \lambda_i(k)} \quad (20)$$

where

$$y_i(k) = \lambda_i(k)\bar{\Phi}(v_i(k)) = \lambda_i(k)\bar{\Phi}(\mathbf{u}_i^{\mathrm{T}}(k)\cdot\mathbf{w}_i(k)) \quad (21)$$

From Eqs. (20) and (21), we compute the partial derivative as

$$\begin{aligned}\frac{\partial y_i(k)}{\partial \lambda_i(k)} &= \bar{\Phi}(\mathbf{u}_i^{\mathrm{T}}(k)\cdot\mathbf{w}_i(k)) + \lambda_i(k)\bar{\Phi}'(\mathbf{u}_i^{\mathrm{T}}(k)\cdot\mathbf{w}_i(k)) \\ &\quad \times \frac{\partial[\mathbf{u}_i^{\mathrm{T}}(k)\cdot\mathbf{w}_i(k)]}{\partial \lambda_i(k)} \\ &= \bar{\Phi}(\mathbf{u}_i^{\mathrm{T}}(k)\cdot\mathbf{w}_i(k)) + \lambda_i(k)\bar{\Phi}'(\mathbf{u}_i^{\mathrm{T}}(k)\cdot\mathbf{w}_i(k)) \\ &\quad \times \frac{\partial}{\partial \lambda_i(k)}\left(\sum_{n=1}^{p} w_{i,n}s(k-n) + w_{i,p+1}(k)\right. \\ &\quad + \sum_{m=1}^{q} w_{i,p+m+1}(k)y_1(k-m) \\ &\quad \left. + \sum_{l=1}^{N} w_{1,p+q+l}(k)y_l(k-l)\right) \end{aligned} \quad (22)$$

Since $\dfrac{\partial \lambda_i(k-1)}{\partial \lambda_i(k)} = 0$

the second term of Eq. (22) vanishes. This way we have

$$\nabla_{\lambda_i(k)} E_i(k) = \frac{\partial E_i(k)}{\partial \lambda_i(k)} = -e_i(k)\bar{\Phi}(v_i(k)) \quad (23)$$

We next consider three cases, that is when all the neurons in the RNN share a common λ, when λ is kept common within a layer and when every neuron has a different λ.

3.1. Case 1: single λ over the whole network

In this case $\lambda_i(k) = \lambda(k)$ for all units for which Eq. (18) holds. Thus we could write

$$y_i(k) = \Phi(v_i(k)) = \lambda(k)\bar{\Phi}(v_i(k)) \quad (24)$$

$$\lambda(k+1) = \lambda(k) - \rho \nabla_{\lambda(k)} E_1(k)$$
$$= \lambda(k) + \rho e_1(k) \bar{\Phi}(v_1(k)) \quad (25)$$

3.2. Case 2: different λ for each layer

In this case, the fully connected recurrent network can be considered to have two layers. The first layer consists of the output neuron and the second layer contains the rest of the neurons in the network. This way

$$y_i(k) = \Phi(v_i(k)) = \begin{cases} \lambda_1(k)\bar{\Phi}(v_i(k)), & i = 1 \\ \lambda_2(k)\bar{\Phi}(v_i(k)), & i = 2,\dots,N \end{cases} \quad (26)$$

$$\begin{aligned} \lambda_1(k+1) &= \lambda_1(k) + \rho e_i(k)\bar{\Phi}(v_i(k)) \\ \lambda_2(k+1) &= \lambda_2(k) + \rho e_i(k)\bar{\Phi}(v_i(k)) \end{aligned} \quad (27)$$

3.3. Case 3: different λ_i for each neuron of the network

This is the most general case where each neuron has it own amplitude adaptation λ. In this case the equation becomes

$$y_i(k) = \Phi(v_i(k)) = \lambda_i(k)\bar{\Phi}(v_i(k)) \quad (28)$$

$$\lambda_i(k+1) = \lambda_i(k) - \rho \nabla_{\lambda_i(k)} E_i(k)$$
$$= \lambda_i(k) + \rho e_i(k)\bar{\Phi}(v_i(k)) \quad (29)$$

4. Simulations

For the experiments, the amplitude of the input signals were scaled to be within the range [0,0.1] and the nonlinearity at the neuron was chosen to be the logistic sigmoid function,

$$\Phi(x) = \frac{\lambda(k)}{1 + e^{-\beta x}} \quad (30)$$

with a slope $\beta = 1$, learning rate $\eta = 0.3$ and an initial amplitude $\lambda(0) = 1$. The step size of the adaptive amplitude was chosen to be $\rho = 0.15$. The architecture of the network for this experiment consists of five neurons with tap input length of 7 and feedback order of 7. The input signal was a AR(4) process given by

$$r(k) = 1.79r(k-1) - 1.85r(k-2) + 1.27r(k-3)$$
$$- 0.41r(k-4) + n(k) \quad (31)$$

with $n(k) \sim \mathcal{N}(0,1)$ as the driving input. The noise was normally distributed $\mathcal{N}(0,1)$. The nonlinear input signal

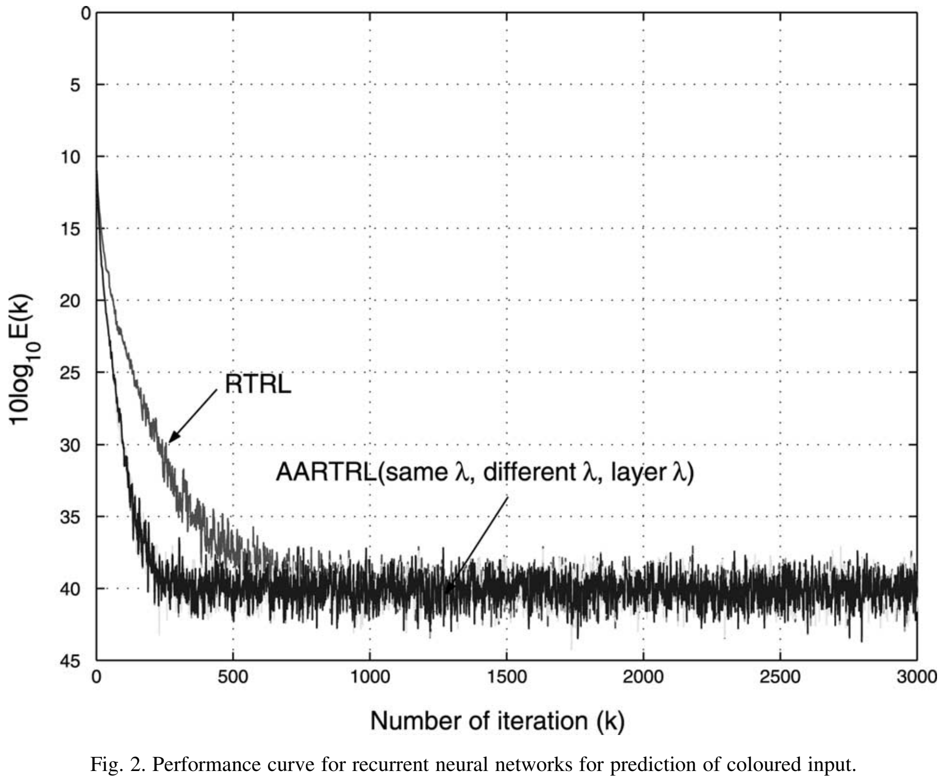

Fig. 2. Performance curve for recurrent neural networks for prediction of coloured input.

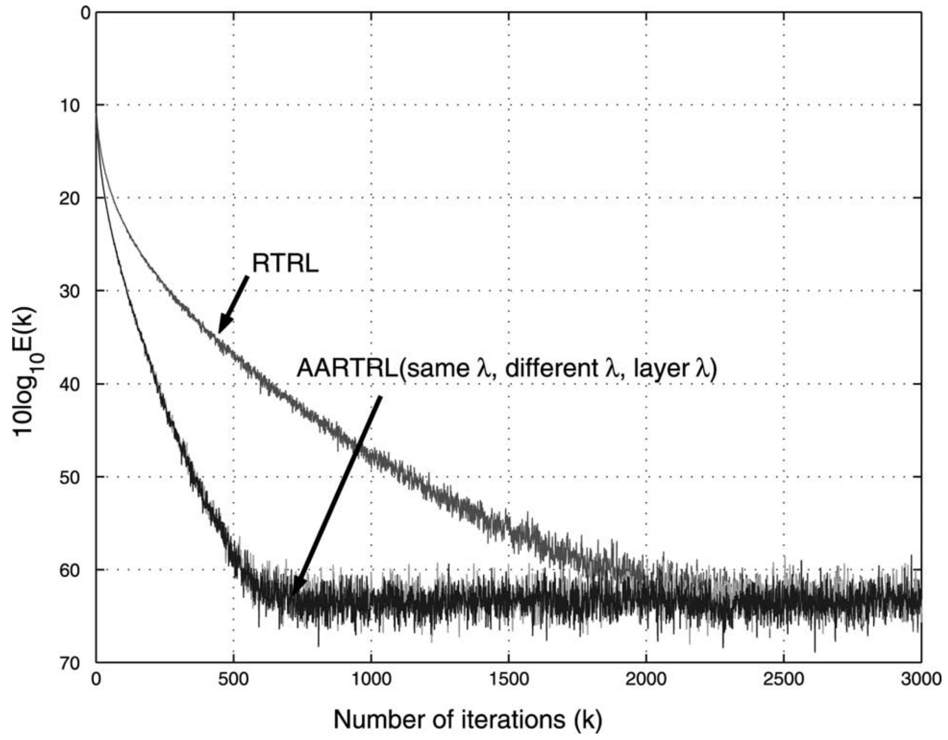

Fig. 3. Performance curve for recurrent neural networks for prediction of nonlinear input.

was calculated as (Narendra & Parthasarathy, 1990)

$$z(k) = \frac{z(k-1)}{(1+z^2(k-1))} + r^3(k) \tag{32}$$

Monte Carlo analysis of 100 iterations was performed on the prediction of the coloured and nonlinear input signals. Figs. 2 and 3 show, respectively, the performance curve of the RTRL and AARTRL algorithm on coloured and nonlinear input signals. For both cases, the AARTRL algorithm outperformed the RTRL algorithm. The AARTRL algorithm gives a faster error convergence. To further investigate the algorithm, the AARTRL algorithm was used to predict signals with rich dynamics such as speech. Fig. 4 shows the plot of the adaptive amplitude, λ.

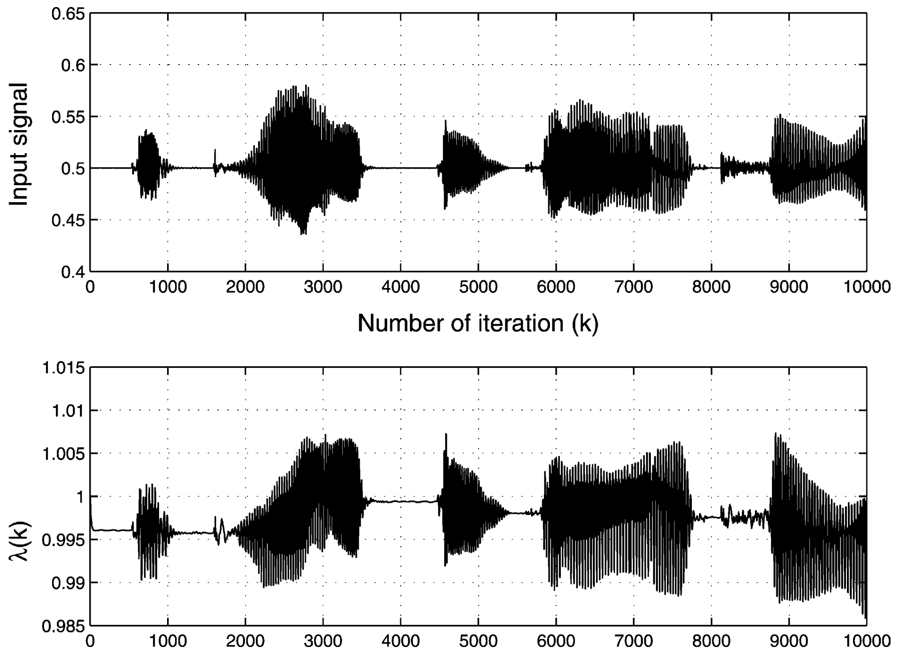

Fig. 4. Adaptive amplitude for AARTRL having the same λ in the whole network on speech signals.

Table 1
Mean Squared Error (MSE) obtained for each model on nonlinear and coloured input

Learning algorithm model	MSE obtained			
	Nonlinear input		Coloured input	
	Average	SD	Average	SD
RTRL with fixed amplitude	0.0146	0.0115	0.0309	0.019
RTRL with single trainable amplitude	0.0118	0.0161	0.0120	0.0172
RTRL with 'layer-by-layer' trainable amplitude	0.0115	0.0182	0.0119	0.0172
RTRL with neuron-by-neuron trainable amplitude	0.0110	0.0177	0.0116	0.0166

It is clearly shown that the amplitude of λ adapts to the dynamics of the signal accordingly. The quantitative results simulations are summarized in Table 1. This table shows the evaluation of the performance of the resulting network given in terms the average value of the cost function $E(k)$. From the experiment, case 3 has the best performance compared to cases 1 and 2. According to Trentin, although case 3 may seem more powerful, allowing for training of individual amplitudes, the resulting algorithm has more parameters meaning more complication and complexity involved.

5. Conclusion

The amplitude of the activation function in the RTRL algorithm has been made adaptive by employing a gradient descent based approach trainable amplitude. The proposed algorithm has been shown to converge faster than the standard RTRL algorithm for nonlinear adaptive prediction.

References

Haykin, S. (1999). *Neural networks: A comprehensive foundation*. Englewood Cliffs, NJ: Prentice-Hall.

Mandic, D. P., & Chambers, J. A. (2001). *Recurrent neural networks for prediction: Learning algorithms, architectures and stability*. London: Wiley.

Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*(1), 4–27.

Trentin, E. (2001). Network with trainable amplitude of activation functions. *Neural Networks*, *14*, 471–493.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280.