Available online at www.sciencedirect.com
SCIENCE DIRECT®
Nonlinear Analysis 63 (2005) e867–e876
Nonlinear Analysis
www.elsevier.com/locate/na

On-line Gauss–Newton-based learning for fully recurrent neural networks

A.A. Vartak[a], M. Georgiopoulos[a], G.C. Anagnostopoulos[b,*]

[a]*Electrical Engineering Department, University of Central Florida, Orlando, FL 32816, USA*
[b]*Electrical & Computer Engineering Department, Florida Institute of Technology, Melbourne, FL 32901, USA*

Abstract

In this paper we propose a novel, Gauss–Newton-based variant of the Real Time Recurrent Learning (RTRL) algorithm by Williams and Zipser (Neural Comput. 1 (1989) 270–280) for on-line training of Fully Recurrent Neural Networks. The new approach stands as a robust and effective compromise between the original, gradient-based RTRL (low computational complexity, slow convergence) and Newton-based variants of RTRL (high computational complexity, fast convergence). By gathering information over time in order to form Gauss–Newton search vectors, the new learning algorithm, GN–RTRL, is capable of converging faster to a better quality solution than the original algorithm. Experimental results reflect these qualities of GN–RTRL, as well as the fact that GN–RTRL may have in practice lower computational cost in comparison, again, to the original RTRL.

1. Introduction

Fully Recurrent Neural Networks (FRNN) are simple, but powerful computational models that can effectively learn temporal sequences, either in an on-line or an off-line fashion. A basic block diagram of an FRNN is shown in Fig. 1. The FRNN consists of a linear input layer and a nonlinear output layer. The input layer is fully connected to the output layer via adjustable, weighted connections, which represent the system's training parameters. The model also features unit-gain, unit-delay feedback connections that are fed back into its

* Corresponding author. 5972 Bent Pine Drive #373, Orlando, FL 32822, USA. Tel.: +1 407 823 5338; fax: +1 407 823 5835.

0362-546X/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.na.2005.02.015

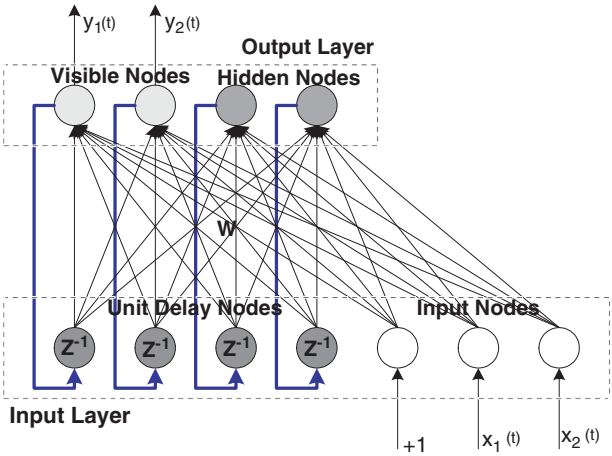

Fig. 1. Block diagram of a Fully Recurrent Neural Network (FRNN).

input layer. FRNNs accomplish their task by learning a mapping between a set of input sequences to another set of output sequences. In particular, the model's inputs consist of input sequences, delayed output activations and a constant-valued input terminal related to a bias weight. On the other hand, the output layer generates the set of output sequences. Typically, nodes in this layer feature a sigmoidal activation function. Furthermore, the model may include a number of hidden nodes, also known as context units, whose activations are not related to any of the outputs of the task to be learned, but act as a secondary, dynamic memory of the system. The combination of this dynamic, context-based memory with the recurrent, feedback connections is what makes the FRNN a powerful model for learning relationships between temporal sequences.

One of the most popular FRNN training schemes is the Real Time Recurrent Learning (RTRL) algorithm [21]. RTRL is a gradient-based algorithm that is used for adjusting the input-to-output interconnection weights. Williams and Zipser present two variations of RTRL, one for off-line (batch) and one for on-line (incremental) learning. In both of its forms, RTRL has been used to train FRNNs for a variety of applications, such as speech recognition and controller modeling. In specific, RTRL has been used for training a robust, manufacturing process controller in [10]. The problem of speech enhancement and recognition is addressed in [11], where RTRL is used to construct adaptive fuzzy filters. RTRL has also been used to train FRNNs for next-symbol prediction in an English text processing application [17]. The RTRL/FRNN combination has also been used in applications of communication systems. Li et al. [12] use FRNNs trained by RTRL for adaptive pre-distortion linearization of RF amplifiers and have shown to attain superior performance in comparison with other well-known pre-distortion models. Furthermore, the authors show significant improvements in the Bit Error Rate (BIR) performance as compared with linear techniques in the field of digital, mobile–radio systems. Finally, RTRL has been used to train FRNNs that were capable of effectively removing artifacts in EEG (Electroencephalograms) signals [19,20].

A plethora of RTRL variants or substitutes have been suggested in the literature that aimed to enhance different aspects of the training procedure such as its computational complexity, especially when used in off-line mode, convergence speed/properties, and its sensitivity to the choice of initial weight values. In [3] a technique is presented for reinitializing RTRL after a specific time interval, so that weight changes depend on fewer past values and weight updates follow more precisely the error gradient. Also, the relationship between some inherent parameters, like the slope of the sigmoidal activation functions and the learning rate, has been taken into account to reduce the degrees of freedom of the associated nonlinear optimization problems [14]. In [18] the gradient calculation has been decomposed into blocks to produce an algorithm which is an order of magnitude faster than the original RTRL. Additional constraints have been imposed to the synaptic weight matrix to achieve reduced learning time, while the network forgetting is reduced [6]. In [4] a conjugate-gradient variation of RTRL has been developed. Other techniques suggested to improve the convergence rate include use of normalized RTRL [15], use of genetic algorithms [13,2]. It has been shown that most of the training approaches are based on different computational ways to efficiently obtain the gradient of error function and can be generally grouped into five major groups. Furthermore, these five approaches are only five different ways of solving particular matrix equation [1]. In the same reference an approach of approximating the gradient was tried, that gave faster convergence. Static grouping of processing elements has also been tried to reduce the computational complexity at the expense of performance [7]. Finally, Newton-based approaches for small FRNNs have also been used to exploit the methods' quadratic rate of convergence [5]. However, the latter approach leads to enormous increase in computational effort and memory allocation needs.

In this paper we present a novel, on-line RTRL variation, namely the GN–RTRL. While the original RTRL training procedure utilizes gradient information to guide the search towards the minimum training error (and therefore we are going to refer to it as GD–RTRL), GN–RTRL uses the Gauss–Newton direction vector for the same purpose. The development of a GN-based training algorithm for FRNNs was motivated by the very nature of the optimization problem at hand. The function to be minimized is of squared-error type, which makes it a Nonlinear Least-Squares (NLS) optimization problem. While gradient descent methods are straightforward and easy to implement for NLS problems, their convergence rate is linear [16], which typically translates to long training times. The problem is further worsened when model size (the number of interconnection weights) increases. On the other side of the spectrum, Newton-based methods attain a theoretical quadratic rate of convergence [16], which makes them appealing from a reduced training time perspective. Nevertheless, Newton-based algorithms require second-order derivative information, either by having available the associated Hessian matrix or approximations to it. The last requirement attaches a high computational cost, which prohibits the usability of Newton-based learning for moderate or large size FRNN structures. We propose an RTRL scheme based on the Gauss–Newton method as a compromise between gradient descent and Newton's methods. The GN–RTRL features a super-linear convergence profile (faster than GD–RTRL) and lower computational cost to second-order algorithms, which makes it practical for training small to moderate size FRNNs.

The rest of the paper is organized as follows. In Section 2 we outline the on-line version of the RTRL algorithm and in Section 3 we present the novel, Gauss–Newton approach for

on-line training of FRNNs. Experimental results comparing the original RTRL and the new training algorithm for two one-step-ahead prediction datasets are presented in Section 4. Finally, in Section 5 we summarize the conclusions of our experimental study.

2. Outline of gradient descent RTRL

We first describe the original, on-line RTRL algorithm (GD–RTRL) in brief. The function to be minimized is the instantaneous sum-of-squared error (SSE) at time instance t, which is displayed in Eq. (1); L is the number of the FRNN's visible outputs.

$$\phi(\boldsymbol{\theta}, t) = \frac{1}{2} \sum_{k=0}^{L-1} e_k^2(t). \tag{1}$$

Vector $\boldsymbol{\theta}$ represents all the FRNN parameters. The error for output k is calculated as follows:

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } \exists d_k(t)\ k = 0 \dots L-1, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

where $d_k(t)$ is the desired output and $y_k(t)$ is the actual output of the network. Notice, that if there is no available desired output for a particular output and/or time instance, the error is considered to be zero. Let H be the number of hidden units in the output layer and $U = L + H$ be the total number of output layer neurons. Additionally, let I be the number of FRNN inputs and $V = U + I + 1$ the total number of input layer neurons. Fig. 1 illustrates the case, where $L = 2$, $H = 2$ and $I = 2$. Then, the collection of those weights is summarized in the $\mathbf{W} \in R^{U \times V}$ matrix. As mentioned before, the elements of the weight matrix $\mathbf{W}$ constitute the parameters of the network. The goal of the training procedure is to adjust $\mathbf{W}$ in order to minimize the instantaneous SSE according to a rule of the form

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \lambda \mathbf{p}, \tag{3}$$

where $\mathbf{p}$ is an appropriate direction vector and λ is the step length in the direction of $\mathbf{p}$ (also referred to as the learning rate). In GD–RTRL this direction vector is the negative gradient of the SSE with respect to the weights:

$$\mathbf{p} = \mathbf{p}^{GD} = -\nabla_{\mathbf{w}} \phi. \tag{4}$$

It is worth noting that in the seminal paper of RTRL no specific guidelines are articulated regarding the choice of the learning rate, apart from the fact that it has to be gradually decreased. The calculation of the gradient goes as follows:

$$\frac{\partial \phi}{\partial w_{i,j}} = -\sum_{k=0}^{U-1} e_k(t)\, p_{i,j}^k(t) \quad i = 0 \dots U-1,\ \ j = 0 \dots V-1, \tag{5}$$

where the output sensitivities p are computed as

$$p_{i,j}^{k}(t) = f'(s_k(t)) \sum_{l=0}^{U-1} w_{k,l+I+1} p_{i,j}^{k}(t-1) + \delta_{k,i} \begin{cases} x_j(t) & \text{if } j = 0 \dots I-1, \\ 1 & \text{if } j = I, \\ y_{j-I-1}(t) & \text{if } j = I+1 \dots V-1. \end{cases} \tag{6}$$

Notice that the output sensitivities p in Eq. (6) are initialized to zero prior to training. The function f is the sigmoidal activation function of the output neurons. Typical choices are the logistic or hyperbolic tangent functions. Additionally,

$$s_k(t) = \sum_{l=0}^{I-1} w_{k,l} x_l(t) + w_{k,I} + \sum_{l=0}^{U-1} w_{k,l+I+1} y_l(t-1), \tag{7}$$

$$y_k(t) = f(s_k(t)). \tag{8}$$

The quantity in Eq. (7) represents the cumulative weighted input to the kth output neuron. A variant to Eq. (7) replaces the actual outputs $y(t-1)$ with the corresponding desired output, a technique referred to as teacher-forced learning [21]. GD–RTRL training starts with a randomized matrix $\mathbf{W}$ and continues with the weight adjustment procedure until the magnitude of the changes becomes small enough, at which point completion of training is declared. In reality, the appropriate value for the learning rate λ has to be identified in an adaptive manner at each step via a line minimization technique in order to achieve convergence.

3. Gauss–Newton RTRL

The presented, new variant of RTRL, the GN–RTRL, replaces the search vector $\mathbf{p}$ in Eq. (3) with the Gauss–Newton direction vector:

$$\mathbf{p} = \mathbf{p}^{GN} = -[\mathbf{J}^{\mathrm{T}}(t)\mathbf{J}(t)]^{-1}\mathbf{J}^{\mathrm{T}}(t)\mathbf{r}(t), \tag{9}$$

where $\mathbf{r}(t) \in \mathbf{R}^{LT}$ is the vector of residuals (errors) and $\mathbf{J}(t) \in \mathbf{R}^{LT \times UV}$ is the Jacobian matrix at time instance t. The former quantity is defined as

$$\mathbf{r}(t) = [e_0(t)e_1(t) \dots e_{L-1}(t)e_0(t-1) \dots e_{L-1}(t-T+1)]^{\mathrm{T}} \tag{10}$$

In other words the residual vector contains errors for different (visible) outputs across T consecutive time instances: the current time instance t and the $T-1$ previous time instances. Notice that the T-exponent in the previous and subsequent equations denotes the transpose of a vector and matrix and should not be confused with the total observed time instances T. If we define

$$\boldsymbol{\theta} \hat{=} vec[\mathbf{W}] \tag{11}$$

then the (m,n) entry of the Jacobian matrix is given as

$$\mathbf{J}(t; m, n) = \frac{\partial r(t)_m}{\partial \theta_n}. \tag{12}$$

The matrix $[\mathbf{J}^{\mathrm{T}}(t)\mathbf{J}(t)]^{-1}\mathbf{J}^{\mathrm{T}}(t)$ in Eq. (9) can be viewed as a low computational cost approximation to the Hessian matrix associated to the SSE minimization, which is sufficiently accurate for small-residual problems [16]. Moreover, it can be shown that

$$\mathbf{J}(t; m, n) = -\, p^k_{i,j}(t), \quad t = \lfloor m/L \rfloor, k = m - Lt, \quad i = \lfloor n/V \rfloor, \quad j = n = Vi \tag{13}$$

which links the Jacobian matrix to the output sensitivities p. Eq. (13) underlines the fact that GN–RTRL is closely related to GD–RTRL, since both approaches utilize output sensitivity information for the computation of their corresponding search directions. Nevertheless, GN–RTRL promises shorter training time without the need of computing second order derivatives. The explicit formation of $[\mathbf{J}^{\mathrm{T}}(t)\mathbf{J}(t)]^{-1}\mathbf{J}^{\mathrm{T}}(t)$ is unnecessary (as it will be shown shortly) and even undesirable, since it is prone to considerable, numerical errors during its computation. Instead, we solve the equation

$$\mathbf{J}(t)\mathbf{p}^{GN} = -\mathbf{r}(t) \tag{14}$$

using a more numerically stable approach, namely Singular Value Decomposition (SVD), as presented in [9]. It is this decomposition which increases, sometimes significantly (especially as model size increases), the computational overhead of GN–RTRL, when compared to GD–RTRL. Solving Eq. (14) this way provides us a solution for the direction vector expressed in terms of the left- ($\mathbf{u}_i$), right-eigenvectors ($\mathbf{v}_i$) and singular values (σ_i) of the Jacobian matrix

$$\mathbf{p}^{GN} = -\sum_i \frac{\mathbf{u}_i^{\mathrm{T}}\mathbf{r}(t)}{\sigma_i}\mathbf{v}_i. \tag{15}$$

In the above summation terms that correspond to relatively small singular values are omitted to increase robustness in the direction vector calculations. This phenomenon may occur when $\mathbf{J}(t)$ is close to being rank deficient and the unaltered search vector may not represent a descent direction. Another alternative would be using the negative gradient as in GN–RTRL for that particular time step until $\mathbf{J}(t)$ becomes full rank again. By virtue of the direction vector's nature, GN–RTRL equipped with a decent line minimization technique is expected to feature super-linear convergence for small-residual problems, when the Jacobian matrix is of full column rank. In an effort to fulfill the last requirement, GN–RTRL must compute its direction vector based on $T \geqslant UV/L$ time instances, which contrasts GD–RTRL that needs only current time instance information. Although this very fact might be considered as a disadvantage of GN–RTRL from a computational or memory-storage perspective, utilizing information of T instead of just one time instance to perform on-line learning may cause a smoothing/averaging effect in time and allow GN–RTRL to converge faster in on-line mode, as it is shown in our experiments.

Table 1
Santa-Fe time series

	TSUC			KFlops			SSE		
	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
GD–RTRL	163	**750.43**	1596	153.01	**710.74**	1517.35	10.86	90.58	1016
GN–RTRL	16	**55.92**	472	13.72	**96.95**	941.41	1.86	109.27	1016

Table 2
Sunspot time series

	TSUC			KFlops			SSE		
	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
GD–RTRL	39	**168.17**	186	36.16	**153.86**	170.37	6.88	35.95	101.44
GN–RTRL	16	**58.48**	1992	19.28	**127.93**	5262.52	1.65	18.08	1013.5

4. Experimental results

In order to demonstrate the merits of the Gauss–Newton RTRL we compared it to the original algorithm on two one-step-ahead prediction problems. For both methods we used the same parabolic interpolation technique for approximate line minimization, the same set of initial weights and the same, maximum learning rate. For each dataset the two algorithms performed training using 100 different initial configurations (weights and number of hidden units). Once the algorithms converged, they were tested on all available data and the produced SSE was measured. The datasets we considered were:

1. *Santa-Fe time series competition dataset*: The dataset consists of a computer-generated, one-dimensional temporal sequence of 500 time instances. [Link: http://www-psych.stanford.edu/∼ andreas/Time-Series/SantaFe.html].
2. *Sunspot dataset*: The dataset consists of a temporal sequence representing the annual, average number of sunspot activity as measured in the interval 1749 to 1915. The sequence is one-dimensional and contains 2000 samples. [Link: http://science.msfc.nasa.gov/ssl/pad/solar/greenwch.htm].

The results obtained are summarized in Tables 1 and 2 illustrated on the next page. TSUC denotes time steps until convergence, while KFlops denotes thousands of floating point operations performed during training. The experimental results for both datasets reflect that GN–RTRL is superior in terms of convergence. In the first dataset GN–RTRL achieved convergence on average in only 7% of the time steps required by GD–RTRL, while in the second one in 35%. In terms of computational effort, GN–RTRL seems to be comparable on average in the worst case (Sunspot series: 128 vs. 154 KFlops).

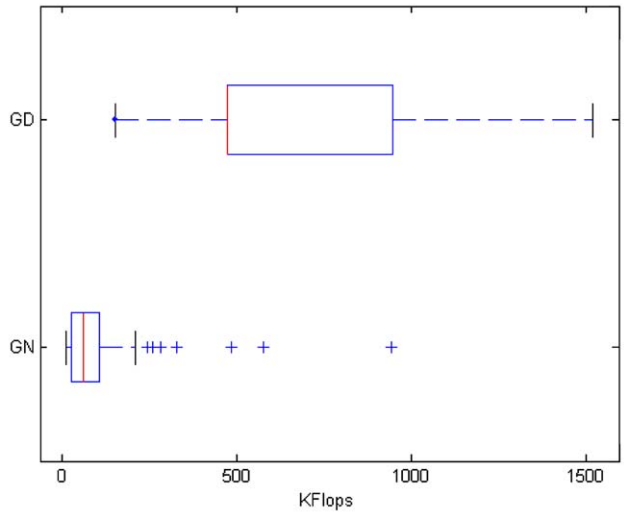

Fig. 2. Boxplot of KFlops for GN–RTRL and GD–RTRL.

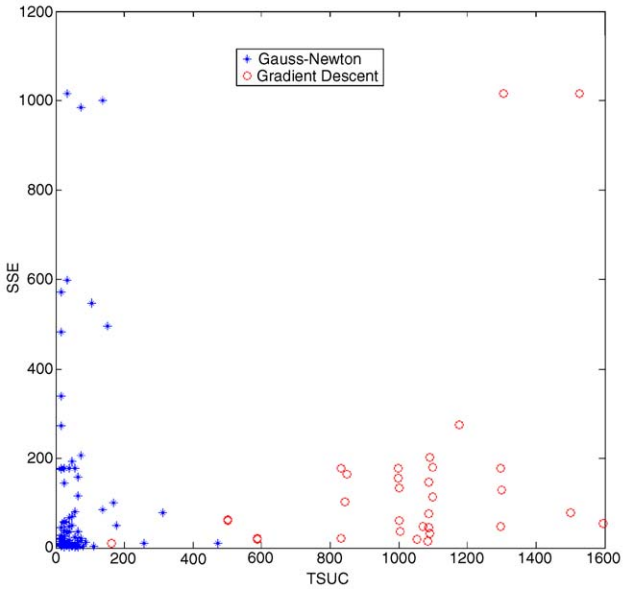

Fig. 3. Plot of performance of GN against GD.

Boxplots describing the distribution of required KFlops for the first dataset is given in Fig. 2. From this plot it can be again concluded that GN–RTRL is relatively more computationally efficient in comparison to GD–RTRL. Despite the fact that GD–RTRL has small computational overhead per time instance, when compared to GN–RTRL that incorporates an SVD step, the latter method compensates by performing significantly less iterations.

In terms of solution quality, Tables 1 and 2 clearly emphasize the superiority of GN–RTRL. The SSE of GN–RTRL is by almost an order of magnitude smaller than the one achieved by GD–RTRL. Fig. 3 shows a representative plot of SSE computed during the testing phase versus the TSUC for both methods, which demonstrates that GN–RTRL converges much faster to a solution and, simultaneously, the solution is of higher quality, when compared to the GD–RTRL experimental results. Additionally, Fig. 3 indicates cases, where it seems that GN–RTRL may have terminated training rather prematurely, which resulted in high SSE values. On the other hand, GD–RTRL exhibits less cases of this sort. This phenomenon is attributed to the near rank deficiency of the Jacobian matrix for certain choices of the random initial weights and can be addressed by taking negative gradient steps, as it has been stated in the previous section.

5. Conclusions

In this paper we presented a Gauss-Newton variation of the Real Time Recurrent Learning algorithm [21] for the on-line training of Fully Recurrent Neural Networks. The modified algorithm, GN–RTRL, performs error minimization using Gauss–Newton direction vectors that are computed from information collected over a period of time rather than only using instantaneous gradient information. GN–RTRL is a robust and effective compromise between the original, gradient-based RTRL (low computational complexity, slow convergence) and Newton-based variants of RTRL (high computational complexity, fast convergence). Experimental results were reported that reflect the superiority of GN–RTRL over the original version in terms of speed of convergence and solution quality. Furthermore, the results indicate that in practice GN–RTRL features a lower-than-expected computational cost due to its fast convergence: GN–RTRL required fewer computations than the original RTRL to accomplish its learning task.

Acknowledgements

The authors acknowledge the partial support from the CCLI-EMD NSF Grant 0341601 and want to thank the two anonymous reviewers for their helpful suggestions and comments.

References

[1] A.F. Atiya, A.G. Parlos, New results on recurrent network training: unifying the algorithms and accelerating convergence, IEEE Trans. Neural Networks 11 (3) (2000) 697–709.
[2] A. Blanco, M. Delgado, M.C. Pegalajar, A real-coded genetic algorithm for training recurrent neural networks, Neural Networks 14 (1) (2001) 93–105.
[3] T. Catfolis, A method for improving the real-time recurrent learning algorithm, Neural Networks 6 (6) (1993) 807–821.
[4] W.F. Chang, M.W. Mak, A conjugate gradient learning algorithm for recurrent neural networks, Neurocomputing 24 (1–3) (1999) 173–189.
[5] P.H.G. Coelho, An extended RTRL training algorithm using Hessian Matrix, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN), vol. 4 (2000) pp. 563–568.

[6] F. Druaux, E. Rogue, A. Faure, Constrained RTRL to reduce learning rate and forgetting phenomenon, Neural Process. Lett. 7 (3) (1998) 161–167.
[7] N. Euliano, J. Principe, Dynamic subgrouping in RTRL provides a faster o(n2) algorithm, Proceedings of the IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP '00), vol. 6, 2000, pp. 3418–3421.
[9] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., John Hopkins University Press, Baltimore, MD, 1996.
[10] A. Hambaba, Robust hybrid architecture to signals from manufacturing and machine monitoring, J. Intell. Fuzzy Syst. 9 (1–2) (2000) 29–41.
[11] C.F. Juang, C.T. Lin, Noisy speech processing by recurrently adaptive fuzzy filters, IEEE Trans. Fuzzy Syst. 9 (1) (2001) 139–152.
[12] C.G. Li, S.B. He, X.F. Liao, J.B. Yu, Using recurrent neural network for adaptive predistortion linearization of RF amplifiers, Int. J. Rf Microwave Comput.- Aided Eng. 12 (1) (2002) 125–130.
[13] M.W. Mak, K.W. Ku, Y.L. Lu, On the improvement of the real time recurrent learning algorithm for recurrent neural networks, Neurocomputing 24 (1–3) (1999) 13–36.
[14] D.P. Mandic, J.A. Chambers, Relating the slope of the activation function and the learning rate within a recurrent neural network, Neural Comput. 11 (5) (1999) 1069–1077.
[15] D.P. Mandic, J.A. Chambers, A normalized real time recurrent learning algorithm, Signal Process. 80 (9) (2000) 1909–1916.
[16] J. Nocedal, S.J. Wright, Numerical Optimization, Springer, New York, 1999.
[17] J.A. Perez-Ortiz, J. Calera-Rubio, M.L. Forcada, Online symbolic-sequence prediction with discrete-time recurrent neural networks, Proceedings of the International Conference on Artificial Neural Networks (ICANN'01), vol. 2130, 2001, pp. 719–724.
[18] J. Schmidhuber, A fixed size storage O(N3) time-complexity learning algorithm for fully recurrent continually running networks, Neural Comput. 4 (2) (1992) 243–248.
[19] S. Selvan, R. Srinivasan, Recurrent neural network based efficient adaptive filtering technique for the removal of ocular artefacts from EEG, IETE Tech. Rev. 17 (1–2) (2000) 73–78.
[20] S. Selvan, R. Srinivasan, Adaptive filtering techniques using neural networks, IETE Tech. Rev. 17 (1–2) (2000) 73–78.
[21] R.J. Williams, D. Zisper, A learning algorithm for continually running fully recurrent neural networks, Neural Comput. 1 (1989) 270–280.