Chapter 3

EFFICIENT SECOND-ORDER LEARNING ALGORITHMS FOR DISCRETE-TIME RECURRENT NEURAL NETWORKS

Eurípedes P. dos Santos and Fernando J. Von Zuben

School of Electrical and Computer Engineering (FEEC)
State University of Campinas (Unicamp)
Brazil

I. INTRODUCTION

Artificial neural networks can be described as computational structures built up from weighted connections among simple and functionally similar nonlinear processing units or nodes, denoted artificial neurons. A class of neural network architectures that has been receiving a great deal of attention in the last few years is that of recurrent neural networks. They can be classified as being globally or partially recurrent.

Globally recurrent neural networks have arbitrary feedback connections, including neurons with self-feedback. On the other hand, partially recurrent neural networks have their main structure non-recurrent, or recurrent but with restrictive properties associated with the feedback connections, like fixed weights or local nature. The presence of feedback allows the generation of internal representations and memory devices, both essential to process spatio-temporal information.

The dynamics presented by a recurrent neural network can be continuous or discrete in time. The analysis of dynamic behavior using already stated theoretical results for continuous dynamic systems and the generation of new results regarding stability of continuos-time recurrent neural networks seem to be the most important appeals to the use of continuous dynamics [Jin, 1995]. However, the simulation of a continuous-time recurrent neural network in digital computational devices requires the adoption of a discrete-time equivalent model.

In this chapter, we will study discrete-time recurrent neural network architectures, implemented by the use of one-step delay operators in the feedback paths. In doing so, digital filters of a desired order can be used to design the network, by a proper definition of connections [Back, 1991]. The resulting nonlinear models for spatio-temporal representation can be directly simulated on a digital computer, by means of a system of nonlinear difference equations. The nature of the equations depends on the kind of recurrent architecture adopted. As a well-known result from signal processing theory, recurrent connections may lead to very complex behaviors, even with a reduced number of parameters and associated equations [Oppenheim, 1999].

Globally and partially recurrent neural networks were shown to perform well in a wide range of applications that involve dynamic and sequential processing [Haykin, 1999]. However, analysis [Kolen, 1994] and synthesis [Cohen, 1992] of recurrent neural networks of practical importance is a very demanding task. As a consequence, the process of weight adjustment in supervised learning is much more demanding in the recurrent case [Williams, 1989], and the availability of recurrent neural networks of practical importance has to be associated with the existence of efficient training algorithms, based on optimization procedures for adjusting the parameters. To improve performance, second-order information should be considered in the training process [Campolucci, 1998, Chang, 1999, Von Zuben, 1995].

So, in what follows, after a brief motivation for using recurrent neural networks and second-order learning algorithms, a low-cost procedure to obtain exact second-order information for a wide range of recurrent neural network architectures will be presented. After that, a very efficient and generic learning algorithm will be described. We will propose an improved version of a scaled conjugate gradient algorithm [Narenda, 1990], that can effectively be used to explore the available second-order information. The original algorithm will be improved based on the detection of important limitations. Basically, we introduce a set of adaptive coefficients to replace fixed ones. These new parameters of the algorithm are automatically adjusted and do not represent additional items to be arbitrarily determined by the user. Finally, some simulation results will be obtained and interpreted.

II. SPATIAL × SPATIO-TEMPORAL PROCESSING

Supervised learning in the context of artificial neural networks can be associated with the use of optimization-based techniques to adjust the network parameters [Poggio, 1990]. The objective is to minimize a cost function, i.e., a function of the input-output data available for learning, that somehow defines the desired behavior to be achieved.

At first glance, neural networks can be divided into two classes: static (non-recurrent) and dynamic (recurrent) networks. Static neural networks are those whose outputs are linear or nonlinear functions of its inputs, and for a given input vector, the network always generates the same output vector. These nets are suitable for processing of spatial patterns. In this case, the relevant information is distributed throughout the spatial coordinates associated with the variables that compose the set of input learning patterns. Typical problems with remarkable spatial dependencies can be found in the areas of pattern recognition and function approximation [Bishop, 1995].

In contrast, dynamic neural networks are capable of implementing memories which gives them the possibility of retaining information to be used later. Now, the network can generate diverse output vectors in response to the same input vector, because the response may also depend on the actual state of the existing memories. By their inherent characteristic of memorizing past information, for long or short-term periods, dynamic networks are good candidates to process patterns with spatio-temporal dependencies, for example, signal processing with emphasis on identification and control of nonlinear dynamic systems [Jin, 1995, Kim, 1997], and nonlinear prediction of time series [Connor, 1994, Von Zuben, 1997].

III. COMPUTATIONAL CAPABILITY

Multilayer perceptron [Haykin, 1999] is a widespread example of a static (non-recurrent) neural network architecture. The main reason it is so effective in worldwide spatial processing applications is the existence of two complementary existential results, with immediate practical effects: a proof of its universal approximation capability [Hornik, 1989] and an effective way to use first- and second-order information, once available, for adjusting the parameters (generally based on the backpropagation algorithm) [Battiti, 1992, van der Smagt, 1994].

However, in the case of recurrent neural network architectures, there are no equivalent *practical* results concerning universal approximation capability with respect to spatio-temporal patterns. A great number of the recurrent neural network architectures, particularly the ones convertible to NARX architectures, share the *existential* property of being capable of simulating Turing machines [Siegelmann, 1997, Siegelmann, 1991], where a Turing machine is an abstraction defined to be functionally as powerful as any computer. However, this very important existential result does not provide any insight about how to achieve the desired behavior, which is why we are faced with so many different architectures to deal with spatio-temporal problems, each one devoted to the specific nature of the problem at hand [Frasconi, 1992, Haykin, 1999].

In spite of some attempts to discover unifying aspects in various architectures [Nerrand, 1993, Tsoi, 1997], the diversity of available architectures that can potentially be applied to solve a given spatio-temporal problem is still commonplace.

In this chapter, we will not try to overcome this troublesome aspect of design. Instead, we will concentrate efforts on developing a generic procedure to obtain first- and second-order information for adjusting the parameters, directly applicable to a wide range of recurrent neural network architectures. Once the first- and second-order information is available, it is important to point out that the same optimization algorithm can be applied, without any kind of modification, to any kind of recurrent (or non-recurrent) neural network architecture.

IV. RECURRENT NEURAL NETWORKS AS NONLINEAR DYNAMIC SYSTEMS

A dynamic system is composed of two parts: the state and the dynamic. The state is formally defined as a multivariate vector of variables, parameterized with respect to time, such that the current value of the state vector summarizes all the information about the past behavior of the system considered necessary to uniquely describe its future behavior, except for the possibly existing external effects produced by inputs applied to the system. The set of possible states is denoted the state space of the system. The dynamic, assumed here to be deterministic, describes how the state evolves through time, and the sequence of states is known as the trajectory in the state space. It is possible to define four classes of dynamic systems [Kolen, 1994], according to the scheme presented in Figure 1.

In essence, the class of recurrent neural networks to be discussed in this chapter is the one characterized by the discrete dynamic and continuous state. The resulting system of difference equations corresponds to a complex nonlinear parametric dynamic system that can exhibit a wide range of behaviors, not produced by static systems.

		State space	
		continuous	discrete
dynamic	continuous	system of differential equations	spin glasses
	discrete	system of difference equations	automata

Figure 1. Classes of dynamic systems

For recurrent neural network with these properties, there are two functional uses [Haykin, 1999]:

- associative memory and
- input-output mapping network.

For example, when globally recurrent neural networks are constrained to have symmetric connections (or some equivalent restrictive property), their asymptotic behaviors are dominated by fixed-point attractors, with guaranteed convergence to stable states from any initial condition. This property can be explored to produce associative memories, as is the case in Hopfield-type neural networks [Li, 1988].

Without such a constraint, the connective structure may assume a wide range of configurations, so that the corresponding recurrent neural network is able to present more complex behaviors than fixed points. The trajectory in the state space will be influenced by a set of attractors and repellers, with arbitrary multiplicity and properly distributed across the state space, each one belonging to one of the following types: fixed point, limit cycle, quasi-periodic, or chaotic [Ott, 1993].

The analysis of computational simulations often considers only the attractors, because the repellers can not be observed. Once the state has reached one attractor, it will stay there indefinitely, unless an external force pushes the state away. When the trajectory of an autonomous system reaches an attractor, we say that the system is in a stationary state. Both continuous and discrete dynamics are able to present the four types of stationary behavior mentioned above.

So, the dynamical complexity of a recurrent neural network can be measured in terms of the number, type, and relative position of attractors and repellers in the state space. Some preliminary results are already available to synthesize, in a closed form, recurrent neural networks in terms of specific position and extension of a reduced number of attractors and repellers in the state space [Cohen, 1992]. However, in the case of complex configurations of attractors and repellers, and when the description of the dynamic system to be synthesized can not be done in terms of attractors and repellers, the only available way of performing the task is by means of supervised learning.

The formalism of attractors and repellers plays an important role in the study of recurrent neural network stability [Haykin, 1999]. In the analysis of dynamic system theory, in addition to stability, controllability and observability are fundamental aspects. If we can control the dynamic behavior of the recurrent neural network, using external inputs if necessary, then we say that the dynamic is controllable. If we can observe the result of the control applied to the network, then we say that the dynamic is observable. Levin and Narendra [Levin, 1993] have presented important results associated with local controllability and local observability of recurrent neural networks.

V. RECURRENT NEURAL NETWORKS AND SECOND-ORDER LEARNING ALGORITHMS

As stated in the previous section, this chapter will treat recurrent neural networks as input-output mapping networks, giving rise to the necessity of establishing an association between the desired input-output behavior and a specific configuration for the neural network parameters (connective configuration). Unfortunately, this association can not be determined a priori or in a closed form. Then, a desired dynamic behavior should be produced by means of an effective learning process (this procedure is also known as dynamic reconstruction [Haykin, 1999]) responsible for discovering this association, which may not be unique.

As supervised learning should be applied to achieve the desired behavior, the success of the task will depend on two conditions:

- the desired behavior must belong to the range of dynamic behaviors that can be produced by the recurrent neural network and
- the supervised learning process must be capable of finding a desired set of parameter values that will give the final connective configuration to the neural network.

Certainly, the most widespread supervised learning mechanisms for neural networks are those using first-order (gradient) information. In this case, the first-order partial derivatives, or sensitivities, associated with some error measure (based on the difference between the network outputs and some target sequences), are computed with respect to the parameters of the network. Later, this available local information related to the error surface is then used to minimize the error.

As widely reported in the literature, despite their widespread use, the gradient-descent method and its variants are characterized by their slow rate of convergence and in some cases, require the arbitrary setting of learning parameters, such as learning rates, before the beginning of the optimization task [Battiti, 1992]. An inadequate choice may raise difficulties or even prevent the success of the adjustment.

Moreover, specifically in the case of recurrent neural networks, there can be at least one hard additional problem that may trap the gradient-based optimization process: the existence of feedback along the processing makes the error surface present highly nonlinear spots [Pearlmutter, 1995]. This characteristic of the error surface is motivated by the possibility of migrating between two qualitatively distinct nonlinear behaviors merely by means of tuning the feedback gain. For example, even small changes in the network parameters, dictated by the learning algorithm itself, may guide the dynamics of the network to change from stable fixed points to unstable ones, which causes a sudden jump in the error measure. Of course, here we are considering an implicit hypothesis that first-order optimization methods do not work properly out of smooth areas in continuous error surfaces.

In general, these undesirable aspects are the main reasons for the poor average performance of first-order learning algorithms. These problems become even more evident in the case of very demanding tasks, where the network behavior must consider simultaneously a great number of correlated specifications (for example, many attractors and repellers). Examples of these kinds of problems are becoming more frequent in system identification and time series prediction tasks.

To improve performance, second-order information should be considered in the training process. One of the most elaborated second-order algorithms for search in multidimensional nonlinear surfaces is the conjugate gradient method, which was proved to be remarkably effective in dealing with general objective functions and is considered among the best general purpose optimization methods presently available.

Given a procedure to obtain second-order information for any kind of recurrent neural network architecture, the learning procedure can be directly applied without any adaptation to the specific context. That is why the first objective of this work is to describe systematic ways of obtaining exact second-order information for a range of recurrent neural network architectures.

In addition to that, the algorithms to be proposed in a coming section present a computational cost (memory usage and processing time) only two times higher than the cost to acquire first-order information.

VI. RECURRENT NEURAL NETWORK ARCHITECTURES

As already stated, the natural way of investigating the dynamic behavior of recurrent neural networks is to consider them as nonlinear dynamic systems. Let a nonlinear discrete-time stationary dynamic system be represented by the state space equations:

$$\begin{cases} \mathbf{x}_p(k+1) = \mathbf{f}_p(\mathbf{x}_p(k), \mathbf{u}(k)) \\ \mathbf{y}_p(k) = \mathbf{h}_p(\mathbf{x}_p(k), \mathbf{u}(k)) \end{cases} \tag{1}$$

where k is the discrete instant of time, $\mathbf{x}_p \in \Re^n$, $\mathbf{u} \in \Re^m$, and $\mathbf{y}_p \in \Re^r$ are the state, input, and output vectors, respectively; $\mathbf{f}_p : \Re^n \times \Re^m \to \Re^n$ and $\mathbf{h}_p : \Re^n \times \Re^m \to \Re^r$ are continuous vector-valued functions representing the state transition mapping and output mapping, respectively. This state space representation is very general and can describe a large range of important nonlinear dynamic systems. Notice that the output equation is a static mapping.

System identification is a fundamental and challenging research area involving nonlinear dynamic systems, and a common approach to identify systems represented by equation (1) is to adopt parameterized models for the unknown maps $\mathbf{f}_p$ and $\mathbf{h}_p$. In this case, there is a growing amount of research about the use of neural networks to model some important subclasses of nonlinear dynamic systems, subsumed within the class of models represented by equation (1). In the literature, attention has been paid to the analysis and synthesis of neural networks models structured in the form of nonlinear auto-regressive moving average (NARMA) models. In this context, there are two main approaches to synthesize NARMA models. The first one assumes that the dynamic behavior of the system output is governed by a finite set of available input-output measurements. Then, an obvious route to modeling is to choose the NARMA model as a feedforward neural network of the form:

$$\mathbf{y}_m(k) = \hat{\mathbf{g}}_m(\mathbf{y}_p(k-1), \cdots, \mathbf{y}_p(k-n), \mathbf{u}(k), \cdots, \mathbf{u}(k-m)) \tag{2}$$

where $\hat{\mathbf{g}}_m$ represents the input-output map performed by the static neural network and $\mathbf{y}_m$ is the output of the model. This is a kind of series-parallel model and presumes a fairly good knowledge of the actual system structure. This scheme of adaptation has been denoted as equation-error approach by the system identification community and is designated as teacher forcing in the neural network parlance. More recently, in view of its peculiar characteristics, Williams [Williams, 1990] coined it as the conservative approach, when related to neural control.

The second approach to construct neural network NARMA models is argued to be used in situations where the use of past input-output information together with a feedforward nonlinear mapping is not able to satisfactorily represent the actual dynamic system. A typical situation is the use of these static neural network NARMA models when the map $\mathbf{h}_p$ in equation (1) has no inverse. In this case, the representation capability of the model can be improved by the use of a recurrent neural network. If the recurrent paths include the outputs, we have a parallel model. As an example, consider the parallel NARMA model given by the following equation:

$$\mathbf{y}_m(k) = \hat{\mathbf{g}}_m(\mathbf{y}_m(k-1), \cdots, \mathbf{y}_m(k-n), \mathbf{u}(k), \cdots, \mathbf{u}(k-m)) \tag{3}$$

Again, $\hat{\mathbf{g}}_m$ represents the feedforward input-output mapping performed by the neural network, but now the outputs always depend on past values of themselves. In this case, the adaptation of the neural network parameters should be realized by a dynamic learning algorithm. When adjusting the model parameters in this way, we are using the output error approach, raised in the system identification area. Some results in the literature have pointed out that parallel models may give improved performance when compared to their series-parallel counterparts, particularly in the case of noisy systems [Shink, 1989]. This improvement occurs because the parallel model prevents the presence of noisy outputs in the composition of the input vector.

In spite of the more powerful representation capabilities associated with parallel models, few results are available in terms of stability analysis, and more effective learning algorithms are required. Because of the aforementioned characteristics, the use of parallel models in tasks related to neural network identification and control is called liberal approach [Williams, 1990].

Figures 2 to 4 show the three most popular recurrent neural network architectures for spatio-temporal processing, where the neural network parameters are left implicit. In Figure 2 we have the globally recurrent neural network architecture (GRNN). In this architecture, the output of each hidden neuron is used to generate the feedback information. If the feedback paths, indicated by the bold arrows, are removed, then a simpler architecture is produced, called local recurrent neural network (LRNN). When the network outputs are the signals used in the feedback loops, as in Figure 3, we have the output-feedback recurrent neural network (OFRNN) or Jordan network. If all the outputs of the existing neurons are used for feedback, the resulting architecture is the most general and is called a fully recurrent neural network (FRNN), as shown in Figure 4.

With different degrees of extension, all these recurrent neural network architectures have attracted the interest of researches [Tsoi, 1994]. A brief look at Figures 3 and 4 can indicate that OFRNN and FRNN are networks that have to be trained following the liberal approach. There are few results concerning the FRNN architecture, because the high flexibility of its dynamic behavior is not so easy to be accessed [Williams, 1990].

On the other hand, the generality of the GRNN architecture and its universal approximation property have been proved [Jin, 1995]. Another important result obtained is that GRNN and OFRNN are equivalent architectures, if their output neurons are linear [Tsoi, 1997].

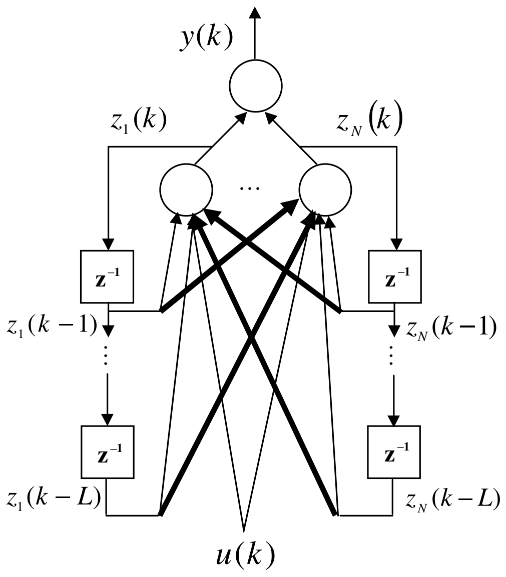

Figure 2. Globally recurrent neural network architecture (GRNN)

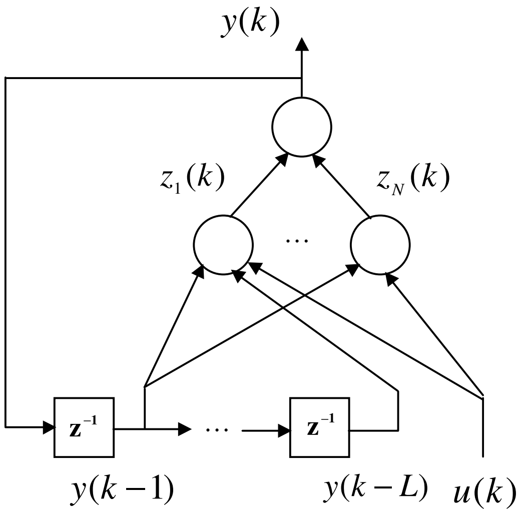

Figure 3. Output-feedback recurrent neural network (OFRNN)

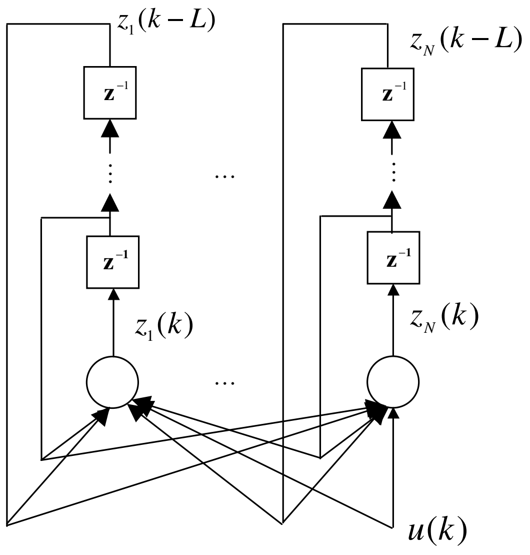

Figure 4. Fully recurrent neural network (FRNN)

VII. STATE SPACE REPRESENTATION FOR RECURRENT NEURAL NETWORKS

The formulations adopted in this work for the learning algorithms are strongly based on matrix manipulations. Hence, in this section a state space representation, valid for each of the three architectures described above, is briefly presented. In the sequel, we take Figures 2 to 4 as guidelines, and we consider that in each architecture there is one hidden layer containing *N* neurons, *M* external inputs, and *O* linear output units. Also, the nonlinear activation function is supposed to be the same for all hidden neurons.

The state variables for any architecture in Figures 2 to 4 can be immediately selected as the set of variables responsible for the memory storage in the recurrent neural network. They are just the past signals available from the tapped delay lines of length L. To make the exposure clear, we can first write the following scalar equations at a particular discrete time instant *k*:

$$s_i(k) = \sum_{j=1}^{D} \sum_{l=1}^{L} a_{il}^{j} x_j(k-l) + \sum_{m=0}^{M} b_{im} u_m(k),\ i = 1,2,\cdots,N \tag{4}$$

$$z_i(k) = f(s_i(k)),\ i = 1,2,\cdots,N \tag{5}$$

$$y_r(k) = \sum_{p=0}^{N} c_{rp} z_p(k), \quad r = 1,2,\cdots,O \tag{6}$$

where the signals $s_i(k)$, $z_i(k)$, and $y_r(k)$ are hidden neurons weighted sums, hidden neurons outputs, and network outputs, respectively. The a_{il}^{j}'s are the weights in the feedback loops, the b_{im}'s are the external inputs gains, and the c_{rp}'s are the weights between the hidden and the output layer. In these equations, $z_0(k)$ and $u_0(k)$ are the bias inputs. The actual values for the state variables x_j's, and the total number of signals to be used to feed the tapped delay lines, *D,* depends on the recurrent network architecture. We have $x_j(t) = z_j(t)$, with $D = N$ for the GRNN and LRNN architectures. Following the same idea, we have $x_j(t) = y_j(t)$, with $D = O$ for the OFRNN. For the FRNN, we also have $x_j(t) = z_j(t)$, with $D = N$. The FRNN network has all its units in a unique layer, and the output neurons (visible neurons) correspond to a subset of these units. Then, the parameters c_{rp} in equation (6) are constants (not adjustable parameters), taking the value 1 if $p=r$ and the value 0 otherwise. In this architecture, the bias inputs for all neurons are removed from the set of output weights and accounted for in the set of input weights.

A matrix formulation for equations (4) to (6) can be obtained as done in what follows:

$$\mathbf{s}(k) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{7}$$

$$\mathbf{z}(k) = \mathbf{F}(\mathbf{s}(k)) = [f(s_1(k)), \cdots, f(s_N(k))]^T \tag{8}$$

$$\mathbf{y}(k) = \mathbf{C}\begin{bmatrix} \mathbf{z}(k) \\ 1 \end{bmatrix} \tag{9}$$

where

$$\mathbf{x}(k) = [x_1(k-1), \cdots, x_1(k-L), \cdots, x_D(k-1), \cdots, x_D(k-L)]^T \tag{10}$$

$$\mathbf{u}(k) = [u_1(k), \cdots, u_M(k), 1]^T \tag{11}$$

$$\mathbf{y}(k) = [y_1(k), \cdots, y_O(k)]^T \tag{12}$$

The entries with value 1 in the vectors appearing in equations (9) and (11) correspond to the bias input. If the network of interest is a FRNN, the bias input must be removed from equation (9). Matrices **A**, **B**, and **C** are formed from the weights a_{il}^{j}'s, b_{im}'s, and c_{rp}'s, respectively, that appear in equations (4) and (6). These matrices have dimensions (*N,D.L*), (*N,M+1*), and (*O,N+1*), respectively.

VIII. SECOND-ORDER INFORMATION IN OPTIMIZATION-BASED LEARNING ALGORITHMS

Supervised learning in an artificial neural network can be formulated as an unconstrained nonlinear optimization problem, where the network parameters are the free independent variables to be adjusted, and an error measure is the dependent variable. The error measure or cost function depends on the network parameters and on the error between the neural network outputs and the desired behaviors dictated by the training examples.

In general, the training examples are in the form of input-output pairs that can be the samples obtained from trajectories generated from some dynamic system, possibly with a nonlinear behavior.

The goal of the supervised learning is to adjust the network parameter so that the trajectories generated by the neural network match the given desired trajectories. Additionally, the trained neural network is required to perform properly when subjected to patterns not seen in the training phase.

Let $\mathbf{w} \in R^{NP}$ be the column vector containing all the neural network weights or adjustable parameters. Also, consider the vectors $\mathbf{y}$ (*k*) and $\mathbf{y}_d$ (*k*), as the neural network output and desired output, respectively. Formally the optimization based learning process is defined by the following equations:

$$\min_{\mathbf{w}} \mathbf{E}_T(\mathbf{w}) \tag{13}$$

$$\mathbf{E_T}(\mathbf{w}) = \sum_{k=1}^{P} \mathbf{E}^k(\mathbf{w}) \tag{14}$$

$$\mathbf{E}^k(\mathbf{w}) = \frac{1}{2}(\mathbf{y}(k) - \mathbf{y}_d(k))^T(\mathbf{y}(k) - \mathbf{y}_d(k)) \tag{15}$$

where *P* is the horizon of time to be considered.

In classical optimization methods, the search for the vector **w** that solves the minimization problem (13) is conducted in an iterative set of steps. In each step, given the actual vector **w,** the optimization procedure can use only information extracted from the cost function to generate a new vector that is a better estimation of the optimal solution.

To see which information is of practical concern, consider the following Taylor series expansion for the error measure around a point **w** in the error surface:

$$\mathbf{E}_\mathrm{T}(\mathbf{w}+\Delta\mathbf{w})=\mathbf{E}_\mathrm{T}(\mathbf{w})+\nabla\mathbf{E}_\mathrm{T}(\mathbf{w})^T\cdot\Delta\mathbf{w}+\frac{1}{2!}\Delta\mathbf{w}^T\cdot\mathbf{H}(\mathbf{w})\cdot\Delta\mathbf{w}+\mathbf{O}\left(\|\Delta\mathbf{w}\|^2\right) \qquad (16)$$

where $\nabla\mathbf{E}_\mathrm{T}(\mathbf{w})$ is the gradient vector with components $\partial\mathbf{E}_\mathrm{T}/\partial w_i$, $\mathbf{H}(\mathbf{w})$ is the Hessian matrix with components $\partial^2\mathbf{E}_\mathrm{T}/\partial w_i\partial w_j$, and $\mathbf{O}\left(\|\Delta\mathbf{w}\|^2\right)$ represent the terms with order higher than two in $\Delta\mathbf{w}$.

Optimization methods that use only objective function evaluation to solve problem (13) form the family of direct search methods. In these methods, some mechanism is used to generate new candidate points and the objective function is used to select the best one. If, at each step, the method uses function evaluations and the first-order information contained in the gradient vector to generate a new point in the search space, it is in the family of steepest descent methods. These methods are characterized by their relative simplicity but slow rate of convergence to a local minimum.

Methods that depend on the information present in the Hessian matrix are second-order methods. These methods work on the hypotheses that a quadratic model is a good local approximation to the objective function. The most representative method in this family is the Newton's method, where, at each step, the inverse of the Hessian matrix is used to generate a new point.

Certainly, the use of higher order information about the error surface can be an effective way of generating improved solutions at each step of the optimization process. It is well known that a second-order method has rate of convergence superior to the one produced by a first-order method [Luenberger, 1989]. But, when dealing with highly nonlinear large-scale optimization problems, practical aspects related with excessive computational burden, numerical errors in matrix operations, and the need for large matrix storage can make such a kind of second-order methods unfeasible to be implemented. In such a situation, even a first-order method is a better choice.

In general, these aspects are frequently present in the supervised neural network learning, where the search space is highly nonlinear and of large dimension. Maybe these are the main reasons for the widespread use of the backpropagation algorithm, in spite of its slowness and oscillatory behavior associated with the use of a fixed learning rate [Jacobs, 1988].

IX. THE CONJUGATE GRADIENT ALGORITHM

Fortunately, all the disadvantages of second-order methods, discussed in the previous section, can be adequately eliminated. To do that, we will employ one of the most effective second-order methods for search in multidimensional nonlinear surface: the conjugate gradient method (CGM). The CGM can be regarded as being somewhat intermediate between the method of steepest descent and Newton's method. It is motivated by the desire to accelerate the typically slow convergence associated with steepest descent, while maintaining simplicity by avoiding the requirements associated with the evaluation, storage, and inversion of the Hessian matrix (or at least the solution of the corresponding system of equations), as required by Newton's method. The storage requirements for the original CGM are for the actual weights, the actual and the immediately previous gradient vector, and two successive search direction vectors.

Originally, the CGM was designed to minimize a quadratic objective function. As an example, consider the quadratic function obtained if the term $\mathbf{O}\left(\|\Delta\mathbf{w}\|^2\right)$ in equation (16) is neglected. Adopting the hypothesis of a quadratic model, the CGM works as follows:

- given two distinct directions $\mathbf{d_1}$ and $\mathbf{d_2}$, they are said to be **H**-orthogonal, or conjugate with respect to a symmetric matrix **H**, if $\mathbf{d}_1^T\mathbf{Hd_2} = \mathbf{d}_2^T\mathbf{Hd}_1 = 0$.
- the CGM is obtained by selecting the successive directions as conjugate with respect to the Hessian matrix. The first direction is set to the current negative gradient vector, and subsequent directions are not specified beforehand but determined sequentially at each step of the iteration.
- the new gradient vector is computed, and linearly combined with previous direction vectors, to obtain a new conjugate direction along which to move.

A. THE ALGORITHM

Initialization: Set random initial values to $\mathbf{w}^o$ and an arbitrarily small value to ε .

Step 1: Starting at $\mathbf{w}^o$, compute $\mathbf{g^o} = \nabla\mathbf{E}_T(\mathbf{w}^o)$ and set $\mathbf{d}^o = -\mathbf{g}^o$.

Step 2: For $k = 0, \Lambda\cdots, NP-1$:

a) Compute $\mathbf{H}(\mathbf{w^j})$;

b) Set $\mathbf{w^{j+1}} = \mathbf{w^j} + \alpha^\mathbf{j}\mathbf{d^j}$, with $\alpha^\mathbf{j} = \dfrac{-(\mathbf{g^j})^T\mathbf{d^j}}{(\mathbf{d^j})^T\mathbf{H}(\mathbf{w^j})\mathbf{d^j}}$; (17)

c) Compute $\mathbf{g^{j+1}} = \nabla\mathbf{E}_T(\mathbf{w^{j+1}})$;

d) Unless $k = NP$-1, set $\mathbf{d^{j+1}} = -\mathbf{g^{j+1}} + \beta^\mathbf{j}\mathbf{d^j}$, (18)

where $\beta^\mathbf{j} = \dfrac{(\mathbf{g^{j+1}})^T\mathbf{H}(\mathbf{w^j})\mathbf{d^j}}{(\mathbf{d^j})^T\mathbf{H}(\mathbf{w^j})\mathbf{d^j}}$. (19)

Step 3: If $\mathbf{E}_T(\mathbf{w}^{NP}) > \varepsilon$, replace $\mathbf{w}^o$ by $\mathbf{w}^{NP}$ and go back to step 1.

For problems where the cost function is exactly quadratic, the Hessian **H** is a constant matrix. It can be proved that in this situation, if **H** is positive definite, the CGM described above converges to the solution in at most *NP* iterations. In the everyday practice, this analytical result may, in some sense, be different due to inevitable numerical errors that are carried over in successive iterations of the method.

B. THE CASE OF NON-QUADRATIC FUNCTIONS

Adopting the already mentioned hypothesis of local quadratic model, the CGM can be extended to general nonlinear objective functions. A nice justification for this assumption is that, near any local optimum, a great variety of nonlinear functions can be well approximated by quadratic functions. This property can be inferred from the Taylor series expansion in equation (16). However, in dealing with general nonlinear functions, the computation of the scalars $\alpha^{\mathbf{j}}$ and $\beta^{\mathbf{j}}$, in equations (17) and (19), requires the calculation of the Hessian matrix at each new point generated by the algorithm. Further, a problem of major concern is that the definiteness property of the Hessian matrix may change from one point to another. It is important to stress the occurrence of the Hessian matrix in the denominator of the expression for the step-length $\alpha^{\mathbf{j}}$. If, at a given point of the search process, the matrix $\mathbf{H}\left(\mathbf{w}^{\mathbf{j}}\right)$ is negative definite, then it is likely that $\alpha^{\mathbf{j}}$ will be negative, resulting in a step along a direction that increases the cost function, instead of decreasing it as expected.

In general, the need for the evaluation of the full Hessian matrix at each new point generated by the CGM is a computational demanding process. This dependence can be suppressed by adopting the following alternatives for the calculation of $\alpha^{\mathbf{j}}$ and $\beta^{\mathbf{j}}$.

The step-length $\alpha^{\mathbf{j}}$ can be obtained by solving the following one-dimensional minimization problem

$$\alpha^{\mathrm{j}} = \arg\min_{\alpha} \mathbf{E}_{\mathrm{T}}\left(\mathbf{w}^{\mathbf{j}} + \alpha\mathbf{d}^{\mathbf{j}}\right) \tag{20}$$

Thus, the value of α used at step *j* is just the one obtained by minimizing the cost function along the line defined by $\mathbf{w}^{\mathbf{j}} + \alpha\mathbf{d}^{\mathbf{j}}$.

Two particular alternative expressions for $\beta^{\mathbf{j}}$, that do not use the Hessian matrix, are of special concern in this work. First, using the definition for $\alpha^{\mathbf{j}}$ and $\beta^{\mathbf{j}}$ given in equations (17) and (19) and the orthogonality property between the gradient at step *j* and all the previous conjugate directions, the following expression can be obtained:

$$\beta_{\mathbf{PR}}^{\mathbf{j}} = \frac{\left(\mathbf{g}^{\mathbf{j+1}}\right)^{T}\left(\mathbf{g}^{\mathbf{j+1}} - \mathbf{g}^{\mathbf{j}}\right)}{\left(\mathbf{g}^{\mathbf{j}}\right)^{T}\mathbf{g}^{\mathbf{j}}} \tag{21}$$

This is known as the Polak-Ribiere expression, and can be further simplified using the orthogonality property between the gradient at step *j* and all the previous gradients, resulting in the Fletcher-Reeves expression:

$$\beta_{\mathbf{FR}}^{\mathbf{j}} = \frac{\left(\mathbf{g}^{\mathbf{j+1}}\right)^T \mathbf{g}^{\mathbf{j+1}}}{\left(\mathbf{g}^{\mathbf{j}}\right)^T \mathbf{g}^{\mathbf{j}}} \qquad (22)$$

If the cost function is exactly quadratic, the two expressions for $\beta^{\mathbf{j}}$, in equations (21) and (22), are equivalent. In the case of more general nonlinear objective functions, the Polak-Ribiere expression is argued to give better results, when compared with the Fletcher-Reeves expression [Johansson, 1992]. This is explained by the fact that, in situations where the algorithm is producing successive points with very little reduction in the objective function, the successive gradient vectors $\mathbf{g}^{\mathbf{j+1}}$ and $\mathbf{g}^{\mathbf{j}}$ are approximately equal in module. Thus, the orthogonality property between gradients is lost, and the Polak-Ribiere expression gives a nearly zero value to $\beta^{\mathbf{j}}$. A small $\beta^{\mathbf{j}}$ has the effect of ruling out the previous search direction and forces a major contribution of the new gradient in the generation of the next search direction, as indicated by the expression in equation (18).

The two alternative expressions described above lead to a CGM that uses only function evaluations and gradient calculations, eliminating the need of the Hessian matrix. But there are some drawbacks associated with the line-search phase necessary to solve problem (20): it is known that the performance of the CGM is sensitive to the accuracy used in the solution of this line-search problem. If the line-search is carried out with great accuracy, the overall performance of the main algorithm will depend on the computations spent on function evaluations used in the line-search phase. On the other hand, a coarse line-search process will produce wrong values for the step-length $\alpha^{\mathbf{j}}$, affecting the orthogonality property between gradients and conjugate directions. Some criteria have been proposed to stop the line-search process when a sufficiently accurate solution for the step-length has been obtained. But these criteria, together with a line-search procedure, introduce problem-dependent parameters that must be specified by the user.

Regardless of the full calculation of the Hessian matrix or the use of line-search procedures, as the algorithm takes its course, the search directions are no longer **H**-conjugate. To alleviate this problem, it is a common practice to reinitialize the direction of search to the negative of the current gradient, after the completion of *NP* iterations. This restart strategy is the simplest one, but more sophisticated strategies can be found in the literature [Bazaraa, 1992].

C. SCALED CONJUGATE GRADIENT ALGORITHM

Moller [Moller, 1993] proposed an effective CGM, called scaled conjugate gradient method (SCGM). In the SCGM, no line search is required and it is considered a procedure to handle the occurrence of negative definite Hessian matrices, at any point in the search space.

The SCGM uses the fact that the Hessian matrix appears in the expression for $\alpha^{\mathbf{j}}$ multiplied by a vector $\mathbf{d}^{\mathbf{j}}$ (see equation (17)). The product of the Hessian

$\mathbf{H}\left(w^k\right)$ by an arbitrary vector $\mathbf{v}$ can be calculated efficiently with the aid of the following finite difference approximation

$$\mathbf{H}(\mathbf{w^k})\mathbf{v} \approx \frac{\nabla\mathbf{E}_T\left(\mathbf{w^k} + \sigma^\mathbf{k}\mathbf{v}\right) - \nabla\mathbf{E}_T\left(\mathbf{w^k}\right)}{\sigma^\mathbf{k}}, \quad 0 < \sigma^\mathbf{k} << 1. \tag{23}$$

In the limit, this approximation tends to the true value of the product $\mathbf{H}\left(\mathbf{w^k}\right)\mathbf{v}$. Here, the trick is to avoid the line-search phase, firstly calculating the approximation to the product $\mathbf{H}\left(\mathbf{w^k}\right)\mathbf{v}$ by equation (23), and then using equation (17) to obtain $\alpha^\mathbf{j}$.

If in some point $\mathbf{w^k}$, the Hessian matrix is negative definite, the use of a possibly negative step-length $\alpha^\mathbf{j}$ is avoided by adding a positive scale parameter λ to the diagonal of $\mathbf{H}\left(\mathbf{w^k}\right)$. If λ is sufficiently large, the Hessian matrix is guaranteed to be positive definite, yielding a positive $\alpha^\mathbf{j}$. Taking a large value for λ implies a small step size in the direction of search $\mathbf{d^j}$, that is, the first-order information will predominate over the second-order information. In a similar way, if the scale parameter λ has a small value, the second-order information will have a major influence than the first-order one in the final value of $\alpha^\mathbf{j}$. To allow the adaptation of λ during the optimization process, the SCGM includes steps inherited from trust region methods that decrease λ in regions where the quadratic model is a good local approximation and increase λ in regions where the quadratic approximation is poor. Detailed description of all the steps in the SCGM can be founded in Moller [Moller, 1993].

X. AN IMPROVED SCGM METHOD

As reported in Moller [Moller, 1993], the SCGM has superior performance when compared with the conventional CGM. In using the original SCGM on highly complex nonlinear surfaces, as those associated with recurrent neural networks, we have observed some problems in the method, regarding the production of negative values for the parameters $\alpha^\mathbf{j}$ and $\beta^\mathbf{j}$. The use of a negative value of $\alpha^\mathbf{j}$, as already stated, indicates that the algorithm is taking a step in a direction that leads to an increase in the objective function. This is a contradictory situation, since we want to minimize the cost function. Also it is known that the convergence of any CGM using the Polak-Ribiere expression for $\beta^\mathbf{j}$ (see equation (21)) is not guaranteed. To alleviate these problems we propose the adoption of a hybridization in the choice of the value to be used for $\beta^\mathbf{j}$. Another important improvement that can be introduced into the SCGM is the exact evaluation of the product $\mathbf{H}(\mathbf{w}^{\mathbf{j}})\mathbf{v}$. At least theoretically, the use of equation (23) is subject to numerical and roundoff problems. In this equation, there are conflicting requirements, as for example the need of small values for $\sigma^{\mathbf{j}}$ in order to obtain a good approximation to the product $\mathbf{H}(\mathbf{w}^{\mathbf{j}})\mathbf{v}$, confronted with precision lost when $\mathbf{v}$ is multiplied by a small value of $\sigma^{\mathbf{j}}$ and used in the sum $\mathbf{w}^{\mathbf{j}} + \sigma^{\mathbf{j}}\mathbf{v}$. Fortunately, the problem related to the exact computation of the product involving the Hessian matrix and an arbitrary vector was entirely solved by Pearlmutter [Pearlmutter, 1994]. Using a differential operator it is possible to compute the product of $\mathbf{H}(\cdot)$ with any desired vector without approximations, and also to avoid the calculation and storage of the Hessian itself.

In the context of recurrent neural networks of practical importance, the application of the SCGM [Moller, 1993], together with the result of Pearlmutter [Pearlmutter, 1994], was firstly considered in Von Zuben and Netto [Von Zuben, 1995] and posteriorly in Campolucci *et al.* [Campolucci, 1998].

A. HYBRIDIZATION IN THE CHOICE OF β^j

It is known that any conjugate gradient method using the Fletcher-Reeves expression (see equation (22)) is globally convergent. The same property can not be guaranteed when using the Polak-Ribiere expression (see equation (21)) [Shewchuk, 1994]. But, as largely reported in the literature, the use of the Polak-Ribiere expression generally leads to superior results [Touati-Ahmed, 1990]. In this chapter, we adopt an idea in some sense similar to one proposed in Touati-Ahmed and Storey [Touati-Ahmed, 1990]. Consider the expressions for $\beta^{\mathbf{j}}_{\mathbf{PR}}$ and $\beta^{\mathbf{j}}_{\mathbf{FR}}$ given in equations (21) and (22), respectively. Our choice for $\beta^{\mathbf{j}}$ is computed as follows:

```
If j = NP-1,
        $\beta^{\mathbf{j}} = \mathbf{0};\ \mathbf{d}^{\mathbf{j+1}} = -\mathbf{g}^{\mathbf{j+1}};\ \mathrm{j} = 0;$
else
        If ( $\beta^{\mathbf{j}}_{\mathbf{PR}}$ >0) and ( $\beta^{\mathbf{j}}_{\mathbf{PR}} < \beta^{\mathbf{j}}_{\mathbf{FR}}$ ),
                $\beta^{\mathbf{j}} = \beta^{\mathbf{j}}_{\mathbf{PR}};$
                $\mathbf{d}^{\mathbf{j+1}} = -\mathbf{g}^{\mathbf{j+1}} + \beta^{\mathbf{j}}\mathbf{d}^{\mathbf{j}};$
                If $-\mathbf{g}^{\mathbf{j+1}}(\mathbf{d}^{\mathbf{j+1}})^T < 0$,
                        $\beta^{\mathbf{j}} = \mathbf{0};\ \mathbf{d}^{\mathbf{j+1}} = -\mathbf{g}^{\mathbf{j+1}};\ \mathrm{j} = 0;$
                else
                        j = j+1;
                end
        else if $\beta^{\mathbf{j}}_{\mathbf{PR}} > \beta^{\mathbf{j}}_{\mathbf{FR}}$,
            $\beta^{j} = \beta^{j}_{FR}$;
            $\mathbf{d}^{j+1} = -\mathbf{g}^{j+1} + \beta^{j}\mathbf{d}^{j}$;
            If $-\mathbf{g}^{j+1}\left(\mathbf{d}^{j+1}\right)^{T} < 0$,
                  $\beta^{j} = \mathbf{0}$; $\mathbf{d}^{j+1} = -\mathbf{g}^{j+1}$; $j = 0$;
            else
                  j = j+1;
            end
      else
            $\beta^{j} = \mathbf{0}$; $\mathbf{d}^{j+1} = -\mathbf{g}^{j+1}$; j =0;
      end
end
```

B. EXACT MULTIPLICATION BY THE HESSIAN [PEARLMUTTER, 1994]

Expanding $\nabla \mathbf{E}_{T}(\cdot)$ around a point $\mathbf{w} \in R^{NP}$ yields:

$$\nabla \mathbf{E}_{T}(\mathbf{w} + \Delta\mathbf{w}) = \nabla \mathbf{E}_{T}(\mathbf{w}) + \mathbf{H}(\mathbf{w}) \cdot \Delta\mathbf{w} + \mathbf{O}\left(\|\Delta\mathbf{w}\|^{2}\right) \tag{24}$$

where $\Delta\mathbf{w}$ is a small perturbation. Choosing $\Delta\mathbf{w} = \alpha\mathbf{v}$, where α is a small real number and $\mathbf{v} \in R^{NP}$ is an arbitrary vector, we can compute **H**(**w**)**v** as follows;

$$\begin{aligned}\mathbf{H}(\mathbf{w})\mathbf{v} &= \frac{1}{\alpha}\left[\nabla \mathbf{E}_{T}(\mathbf{w} + \alpha\mathbf{v}) - \nabla \mathbf{E}_{T}(\mathbf{w}) + \mathbf{O}\left(\alpha^{2}\right)\right] = \\ &\frac{\nabla \mathbf{E}_{T}(\mathbf{w} + \alpha\mathbf{v}) - \nabla \mathbf{E}_{T}(\mathbf{w})}{\alpha} + \mathbf{O}\left(\alpha^{2}\right)\end{aligned} \tag{25}$$

Taking the limit as $\alpha \to 0$,

$$\mathbf{H}(\mathbf{w})\mathbf{v} = \lim_{\alpha \to 0} \frac{\nabla \mathbf{E}_{T}(\mathbf{w} + \alpha\mathbf{v}) - \nabla \mathbf{E}_{T}(\mathbf{w})}{\alpha} = \frac{\partial}{\partial \alpha} \nabla \mathbf{E}_{T}(\mathbf{w} + \alpha\mathbf{v})\Big|_{\alpha=0} \tag{26}$$

Now, it is necessary to introduce a transformation to convert an algorithm that computes the gradient of the system into one that computes the expression in equation (26). Defining the operator

$$\Psi_{\mathbf{v}}\{f(\mathbf{w})\} \equiv \frac{\partial}{\partial \alpha} f(\mathbf{w} + \alpha\mathbf{v})\Big|_{\alpha=0} \tag{27}$$

we have $\Psi_{\mathbf{v}}\{\nabla \mathbf{E}_{\mathrm{T}}(\mathbf{w})\} = \mathbf{H}(\mathbf{w})\mathbf{v}$ and $\Psi_{\mathbf{v}}\{\mathbf{w}\} = \mathbf{v}$. Because $\Psi_{\mathbf{v}}\{\cdot\}$ is a differential operator, it obeys the usual rules of differentiation.

XI. THE LEARNING ALGORITHM FOR RECURRENT NEURAL NETWORKS

To apply the improved SCGM to recurrent neural network learning, we need to compute $\nabla \mathbf{E}_{\mathrm{T}}(\mathbf{w})$ and the product **H**(**w**)**v** for each step *j*. Consider again the recurrent neural network architectures presented in section 6. Let **a**, **b**, and **c** be the column vectors obtained from piling the lines of the matrices **A**, **B,** and **C,** respectively. The ordering in which the lines are taken to form the piles may be arbitrary, as long as the favored order is always adopted from then on. Thus, following the dimensions adopted for the architectures, we can write $\mathbf{a} \in \mathbf{R}^{(\mathbf{N.D.L})}$, $\mathbf{b} \in \mathbf{R}^{(\mathbf{N.(M+1)})}$, and $\mathbf{c} \in \mathbf{R}^{(\mathbf{O.(N+1)})}$. Remember that, if the architecture is a FRNN, matrix **C** and its corresponding vector **c** do not have adjustable parameters.

Now, the vector $\mathbf{w} \in \mathbf{R}^{\mathbf{NP}}$, that contains all the weights of a particular architecture, can be expressed as $\mathbf{w} = \left[\mathbf{a}^{\mathbf{T}} \ \mathbf{b}^{\mathbf{T}} \ \mathbf{c}^{\mathbf{T}}\right]^{\mathbf{T}}$ and has a total number of parameters given by **NP**=(N.D.L) + (N.(M+1)) + O.(N+1).

Given the gradient vector $\nabla \mathbf{E}_{\mathrm{T}}(\mathbf{w})$ of the error measure defined in equation (14), its decomposition to produce the partial gradient vectors with respect to **a**, **b**, and **c,** are columns vectors denoted by $\nabla \mathbf{E}_{\mathbf{T}}^{\mathrm{a}}(\mathbf{w})$, $\nabla \mathbf{E}_{\mathbf{T}}^{\mathbf{b}}(\mathbf{w})$, and $\nabla \mathbf{E}_{\mathbf{T}}^{\mathbf{c}}(\mathbf{w})$, respectively. Following the same notation adopted in the formation of **w**, the vector $\nabla \mathbf{E}_{\mathrm{T}}(\mathbf{w})$ can now be expressed as

$$\nabla \mathbf{E}_{\mathrm{T}}(\mathbf{w}) = \left[\left(\nabla \mathbf{E}_{\mathbf{T}}^{\mathbf{a}}\right)^{\mathbf{T}} \ \left(\nabla \mathbf{E}_{\mathbf{T}}^{\mathbf{b}}\right)^{\mathbf{T}} \ \left(\nabla \mathbf{E}_{\mathbf{T}}^{\mathbf{c}}\right)^{\mathbf{T}}\right]^{\mathrm{T}}.$$

The vector **v**, considered in the calculus of the product **H**(**w**)**v**, has the same dimension of **w**. Actually, **v** will always be taken as the search direction $\mathbf{d}^{\mathbf{j}}$, to be defined at each iteration of the improved SCGM. Thus, **v** can be used to generate three matrices, $\mathbf{V}_{\mathbf{a}}$, $\mathbf{V}_{\mathbf{b}}$, and $\mathbf{V}_{\mathbf{c}}$, with the same dimensions as the matrices **A**, **B**, and **C**, respectively. The process used to distribute the elements of **v** into the matrices $\mathbf{V}_{\mathbf{a}}$, $\mathbf{V}_{\mathbf{b}}$, and $\mathbf{V}_{\mathbf{c}}$ must be the inverse of the one adopted to form **w**.

To help in further developments, in what follows we will define generic vectors and matrices. Considering the column vectors $\Phi = [\phi_1, \cdots, \phi_H]^{\mathbf{T}}$ and $\mathbf{g} = [g_1(\Phi), \cdots, g_P(\Phi)]^{\mathbf{T}}$, we define the following matrices:

$$\Lambda(\mathbf{g}) = \text{block diagonal}\{g_i, i = 1,\cdots,P\} = \begin{bmatrix} g_1 & 0 & 0\cdots & 0 \\ 0 & g_2 & 0\cdots & 0 \\ . & . & . & . \\ 0 & 0 & 0\cdots & g_P \end{bmatrix}$$

$$\Pi(\Phi) = \text{block diagonal}\{[\phi_1,\cdots,\phi_H]\} = \begin{bmatrix} \phi_1\cdots\phi_H & 0\cdots & & 0 \\ 0\cdots 0 & \phi_1\cdots\phi_H & 0\cdots & 0 \\ . & . & . & . \\ 0\cdots & & 0 & \phi_1\cdots\phi_H \end{bmatrix}$$

$\mathbf{J}_\Phi(\mathbf{g})$ = Jacobian matrix of **g** with respect to Φ :

$$\mathbf{J}_\Phi(\mathbf{g}) = \begin{bmatrix} \frac{\partial g_1}{\partial\phi_1} & \frac{\partial g_1}{\partial\phi_2} & \cdots & \frac{\partial g_1}{\partial\phi_H} \\ \frac{\partial g_2}{\partial\phi_1} & \frac{\partial g_2}{\partial\phi_2} & \cdots & \frac{\partial g_2}{\partial\phi_H} \\ . & . & . & . \\ \frac{\partial g_P}{\partial\phi_1} & \frac{\partial g_P}{\partial\phi_2} & \cdots & \frac{\partial g_P}{\partial\phi_H} \end{bmatrix}$$

$\overline{\mathbf{C}}$, $\overline{\mathbf{V}}_\mathbf{c}$: matrices obtained from **C,** and $\mathbf{V_c}$, respectively, by removing the last column. In **C**, this column contains the bias of the output units.

In the following development the equations for the derivative of the error, measured with respect to the network parameters, will be presented through matrix manipulations.

A. COMPUTATION OF $\nabla\mathbf{E_T}(\mathbf{w})$

The operator ∇ is a linear differential operator, and its application to equation (14) results in the following equation:

$$\nabla\mathbf{E_T}(\mathbf{w}) = \sum_{k=1}^{P} \nabla\mathbf{E}^k(\mathbf{w}) \tag{28}$$

Since we are adopting batch learning, the network parameters are updated only after the presentation of all the training patterns. In this case, the total gradient is the sum of the partial gradients calculated at each time step *k*. In the sequel, we will present the equations for the calculation of partial gradients. Each partial gradient can also be broken into its components, corresponding to vectors **a**, **b**, and **c**. Thus, we can write:

$$\nabla \mathbf{E}^{\mathbf{k}}(\mathbf{w}) = \left[\left(\nabla \mathbf{E}_{\mathbf{a}}^{\mathbf{k}}\right)^{\mathbf{T}} \ \left(\nabla \mathbf{E}_{\mathbf{b}}^{\mathbf{k}}\right)^{\mathbf{T}} \ \left(\nabla \mathbf{E}_{\mathbf{c}}^{\mathbf{k}}\right)^{\mathbf{T}} \right]^{\mathrm{T}} \tag{29}$$

Using the definition for $\mathbf{E}^{\mathbf{k}}(\mathbf{w})$, given in equation (15), and the state space representation, given in equations (7) to (12), the following equations can be written:

$$\mathbf{J_c}(\mathbf{z}(k)) = \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\mathbf{A}\mathbf{J_c}(\mathbf{x}(k)) \tag{30}$$

$$\mathbf{J_c}(\mathbf{y}(k)) = \overline{\mathbf{C}}\mathbf{J_c}(\mathbf{z}(k)) + \Pi\left(\begin{bmatrix}\mathbf{z}(k)\\ 1\end{bmatrix}\right) \tag{31}$$

$$\left[\nabla \mathbf{E}_{\mathbf{c}}^{\mathbf{k}}\right]^{\mathbf{T}} = [\mathbf{y}(k) - \mathbf{y}_d(k)]^{\mathbf{T}}\mathbf{J_c}(\mathbf{y}(k)) \tag{32}$$

$$\mathbf{J_a}(\mathbf{z}(k)) = \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\{\mathbf{A}\mathbf{J_a}(\mathbf{x}(k)) + \Pi(\mathbf{x}(k))\} \tag{33}$$

$$\mathbf{J_a}(\mathbf{y}(k)) = \overline{\mathbf{C}}\mathbf{J_a}(\mathbf{z}(k)) \tag{34}$$

$$\left[\nabla \mathbf{E}_{\mathbf{a}}^{\mathbf{k}}\right]^{\mathbf{T}} = [\mathbf{y}(k) - \mathbf{y}_d(k)]^{\mathbf{T}}\mathbf{J_a}(\mathbf{y}(k)) \tag{35}$$

$$\mathbf{J_b}(\mathbf{z}(k)) = \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\left\{\mathbf{A}\mathbf{J_b}(\mathbf{x}(k)) + \Pi\left(\begin{bmatrix}\mathbf{u}(k)\\ 1\end{bmatrix}\right)\right\} \tag{36}$$

$$\mathbf{J_b}(\mathbf{y}(k)) = \overline{\mathbf{C}}\mathbf{J_b}(\mathbf{z}(k)) \tag{37}$$

$$\left[\nabla \mathbf{E}_{\mathbf{b}}^{\mathbf{k}}\right]^{\mathbf{T}} = [\mathbf{y}(k) - \mathbf{y}_d(k)]^{\mathbf{T}}\mathbf{J_b}(\mathbf{y}(k)) \tag{38}$$

B. COMPUTATION OF H(w)v

Given a vector **v**, with the properties already mentioned, the computation of $\mathbf{H}(\mathbf{w})\mathbf{v} = \Psi_{\mathbf{v}}\{\nabla \mathbf{E_T}(\mathbf{w})\}$ requires the application of the derivative operator $\Psi_{\mathbf{v}}\{\cdot\}$ to every calculation done to obtain $\nabla \mathbf{E_T}(\mathbf{w})$. Applying $\Psi_{\mathbf{v}}\{\}$ to equations (7) to (9), we get

$$\Psi_{\mathbf{v}}\{\mathbf{s}(k)\} = \mathbf{A}\Psi_{\mathbf{v}}\{\mathbf{x}(k)\} + \mathbf{V_a}\mathbf{x}(k) + \mathbf{B}\Psi_{\mathbf{v}}\{\mathbf{u}(k)\} + \mathbf{V_b}\mathbf{u}(k) \tag{39}$$

$$\Psi_{\mathbf{v}}\{\mathbf{z}(k)\} = \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\Psi_{\mathbf{v}}\{\mathbf{s}(k)\} \tag{40}$$

$$\Psi_{\mathbf{v}}\{\mathbf{y}(k)\} = \mathbf{C}\begin{bmatrix}\Psi_{\mathbf{v}}\{\mathbf{z}(k)\}\\ 0\end{bmatrix} + \mathbf{V_c}\begin{bmatrix}\mathbf{z}(k)\\ 1\end{bmatrix} \tag{41}$$

Now, applying the operator $\Psi_{\mathbf{v}}\{\cdot\}$ to equation (28) results in the following equation:

$$\Psi_{\mathbf{v}}\{\nabla\mathbf{E_T}(\mathbf{w})\} = \sum_{k=1}^{P}\Psi_{\mathbf{v}}\left\{\nabla\mathbf{E}^{k}(\mathbf{w})\right\} \tag{42}$$

This equation leads to the conclusion that the total product **H**(**w**)**v** can be computed by adding the results of applying the operator $\Psi_{\mathbf{v}}\{\cdot\}$ to each partial gradient computed at time step *k*. Following these guidelines, we obtain:

$$\begin{aligned}\Psi_{\mathbf{v}}\{\mathbf{J_c}(\mathbf{z}(k))\} = {} & \Lambda\left\{\Lambda\left[\ddot{\mathbf{F}}(\mathbf{s}(k))\right]\Psi_{\mathbf{v}}(\mathbf{s}(k))\right\}\mathbf{A}\mathbf{J_c}(\mathbf{x}(k))\\ & + \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\left[\mathbf{A}\Psi_{\mathbf{v}}\{\mathbf{J_c}(\mathbf{x}(k))\} + \mathbf{V_a}\mathbf{J_c}(\mathbf{x}(k))\right]\end{aligned} \tag{43}$$

$$\Psi_{\mathbf{v}}\{\mathbf{J_c}(\mathbf{y}(k))\} = \overline{\mathbf{C}}\Psi_{\mathbf{v}}\{\mathbf{J_c}(\mathbf{z}(k))\} + \overline{\mathbf{V}}_{\mathbf{c}}\mathbf{J_c}(\mathbf{z}(k)) + \Pi\left(\begin{bmatrix}\Psi_{\mathbf{v}}\{\mathbf{z}(k)\}\\ 0\end{bmatrix}\right) \tag{44}$$

$$\Psi_{\mathbf{v}}\left\{\left[\nabla\mathbf{E_c^k}\right]^{\mathbf{T}}\right\} = \left[\mathbf{y}(k) - \mathbf{y}_d(k)\right]^{\mathbf{T}}\Psi_{\mathbf{v}}\{\mathbf{J_c}(\mathbf{y}(k))\} + \Psi_{\mathbf{v}}\{\mathbf{y}(k)\}\mathbf{J_c}(\mathbf{y}(k)) \tag{45}$$

$$\begin{aligned}\Psi_{\mathbf{v}}\{\mathbf{J_a}(\mathbf{z}(k))\} = {} & \Lambda\left\{\Lambda\left[\ddot{\mathbf{F}}(\mathbf{s}(k))\right]\Psi_{\mathbf{v}}(\mathbf{s}(k))\right\}\left[\mathbf{A}\mathbf{J_a}(\mathbf{x}(k)) + \Pi(\mathbf{x}(k))\right\}\\ & + \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\left[\mathbf{A}\Psi_{\mathbf{v}}\{\mathbf{J_a}(\mathbf{x}(k))\} + \mathbf{V_a}\mathbf{J_a}(\mathbf{x}(k)) + \Pi(\Psi_{\mathbf{v}}\{\mathbf{x}(k)\})\right]\end{aligned} \tag{46}$$

$$\Psi_{\mathbf{v}}\{\mathbf{J_a}(\mathbf{y}(k))\} = \overline{\mathbf{C}}\Psi_{\mathbf{v}}\{\mathbf{J_a}(\mathbf{z}(k))\} + \overline{\mathbf{V}}_{\mathbf{c}}\mathbf{J_a}(\mathbf{z}(k)) \tag{47}$$

$$\Psi_{\mathbf{v}}\left\{\left[\nabla\mathbf{E_a^k}\right]^{\mathbf{T}}\right\} = \left[\mathbf{y}(k) - \mathbf{y}_d(k)\right]^{\mathbf{T}}\Psi_{\mathbf{v}}\{\mathbf{J_a}(\mathbf{y}(k))\} + \Psi_{\mathbf{v}}\{\mathbf{y}(k)\}\mathbf{J_a}(\mathbf{y}(k)) \tag{48}$$

$$\begin{aligned}\Psi_{\mathbf{v}}\{\mathbf{J_b}(\mathbf{z}(k))\} = {} & \Lambda\left\{\Lambda\left[\ddot{\mathbf{F}}(\mathbf{s}(k))\right]\Psi_{\mathbf{v}}(\mathbf{s}(k))\right\}\left[\mathbf{A}\mathbf{J_b}(\mathbf{x}(k)) + \Pi\left(\begin{bmatrix}\mathbf{u}(k)\\ 1\end{bmatrix}\right)\right\}\\ & + \Lambda\left(\dot{\mathbf{F}}(\mathbf{s}(k))\right)\begin{bmatrix}\mathbf{A}\Psi_{\mathbf{v}}\{\mathbf{J_b}(\mathbf{x}(k))\} + \mathbf{V_a}\mathbf{J_b}(\mathbf{x}(k))\\ + \Pi\left(\begin{bmatrix}\Psi_{\mathbf{v}}\{\mathbf{u}(k)\}\\ 0\end{bmatrix}\right)\end{bmatrix}\end{aligned} \tag{49}$$

$$\Psi_{\mathbf{v}}\{\mathbf{J_b}(\mathbf{y}(k))\} = \overline{\mathbf{C}}\Psi_{\mathbf{v}}\{\mathbf{J_b}(\mathbf{z}(k))\} + \overline{\mathbf{V}}_{\mathbf{c}}\mathbf{J_b}(\mathbf{z}(k)) \tag{50}$$

$$\Psi_{\mathbf{v}}\left\{\left[\nabla \mathbf{E}_{\mathbf{b}}^{\mathbf{k}}\right]^{\mathbf{T}}\right\} = \left[\mathbf{y}(k) - \mathbf{y}_d(k)\right]^{\mathbf{T}} \Psi_{\mathbf{v}}\{\mathbf{J_b}(\mathbf{y}(k))\} + \Psi_{\mathbf{v}}\{\mathbf{y}(k)\}\mathbf{J_b}(\mathbf{y}(k)) \qquad (51)$$

XII. SIMULATION RESULTS

To show the gain in performance obtained with the proposed hybrid SCGM, we establish a comparison with the original SCGM with exact second-order information, considered to be the best second-order algorithm already proposed in the literature. We take two examples: one concerning nonlinear system identification, and the other a time series prediction. In simulations involving recurrent neural networks with the same architectures, the competing algorithms were initialized with the same set of weights. In all situations, the weights were generated from a symmetric uniform distribution in the range [-0.2,0.2].

The error criterion used in all the simulations is that indicated in equation (14). The network parameters were adapted by presenting the patterns in a batch (epoch-wise) mode. As the main objective is to access the convergence aspects of both versions of the SCGM, major attention is given to the error curves in the learning process.

Nonlinear System Identification: The nonlinear plant used in the generation of the training patterns is the same used in Example 3 of Narendra and Parthasarathy [Narenda, 1990]. The training set consists of 1000 samples of input-output pairs, generated according to the guidelines adopted there. The neural network identifiers receive $u(k)$ as input and have $y_p(k+1)$ as desired output. We carried out simulations with the three recurrent architectures. To exemplify, Figure 5 shows the errors curves obtained in the training of a neural net with the OFRNN architecture. We adopted 5 (five) hidden neurons and tapped delay lines of length $L = 5$. The curve with solid line corresponds to the hybrid SCGM, and the curve with dotted line corresponds to the conventional one. This figure shows that the hybrid SCGM reached the local minimum of the error surface in a reduced number of epochs when compared with the original SCGM.

Time Series Prediction: In this task, we take as training patterns 1000 points of a time series generated from the Lorentz equations, with the same conditions described in Ergezinger and Thomsen [Ergezinger, 1995]. In Figure 6, we show the curves of the error measure for the same recurrent network with the FRNN architecture. The net has 10 (ten) hidden neurons and tapped delay lines of length $L = 1$. Again the hybrid SCGM (solid line) takes advantage over the original SCGM (doted line).

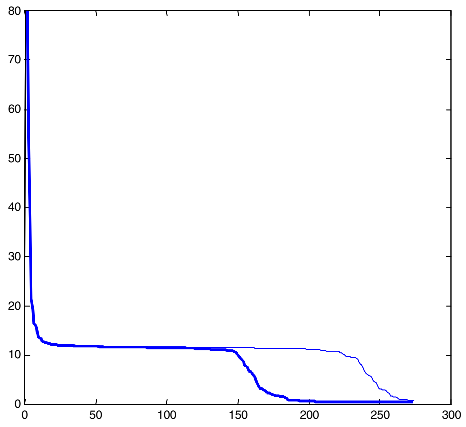

Figure 5. Performance in a system identification task

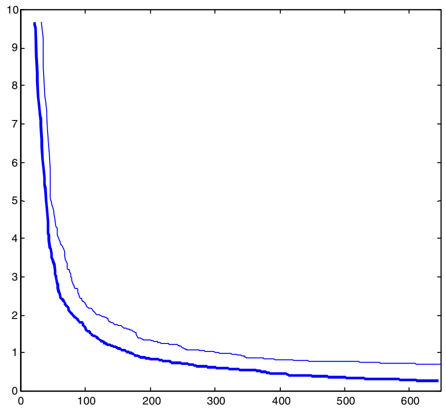

Figure 6. Performance in a time series prediction task

XIII. CONCLUDING REMARKS

Based on the results presented above, we state that globally and partially recurrent neural networks can be applied to represent complex dynamic behaviors. This chapter investigated input-output mapping networks, so that the desired dynamic behavior has to be produced by means of an effective supervised learning process.

The innovative aspects of this work are the proposition of a systematic procedure to obtain exact second-order information for a range of different recurrent neural network architectures, at a low computational cost, and an improved version of a scaled conjugate gradient algorithm to make use of this high-quality information. An important aspect is that, given the exact second-order information, the learning algorithm can be directly applied, without any kind of adaptation to the specific context.

ACKNOWLEDGMENTS

Eurípedes P. dos Santos acknowledges CAPES (DS-44/97-0), and Fernando J. Von Zuben CNPq (300910/96-7) and FAPESP (98/09939-6), for their support.

REFERENCES

Back, A. D. and Tsoi. A. C., FIR and IIR synapses, a new neural network architecture for time series modeling, *Neural Computation*, 3: 375, 1991.

Battiti, R., First- and second-order methods for learning: between steepest descent and Newton's method, *Neural Computation*, 4(2), 141, 1992.

Bazaraa, M. S., Sherali, H. D., and Shetty, C. M., *Nonlinear Programming: Theory and Algorithms*, John Wiley & Sons, New York, 1992.

Bishop, C. M., *Neural Networks for Pattern Recognition*. Oxford Univ. Press, New York, 1995.

Campolucci, P., Simonetti, M., Uncini, A., and Piazza, F., New second-order algorithms for recurrent neural networks based on conjugate gradient, *IEEE International Joint Conference on Neural Networks*, 384, 1998.

Chang, W.-F. and Mak, M.-W., A conjugate gradient learning algorithm for recurrent neural networks, *Neurocomputing*, 24, 173, 1999.

Cohen, M. A., The construction of arbitrary stable dynamics in nonlinear neural networks, *Neural Networks*, 5(1), 83, 1992.

Connor, J. T., Martin, R. D., and Atlas, L. E., Recurrent neural networks and robust time series prediction, *IEEE Transactions on Neural Networks*, 5(2), 240, 1994.

Ergezinger, S. and Thomsen, E., An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer, *IEEE Transactions on Neural Networks*, 6(1), 31, 1995.

Frasconi, P., Gori, M., and Soda, G., Local feedback multilayered networks, *Neural Computation*, 4, 121, 1992.

Haykin, S., *Neural Networks – A Comprehensive Foundation*. Prentice Hall, Englewood Cliffs, NJ, 1999.

Hornik, K., Stinchcombe, M., and White, H., Multi-layer feedforward networks are universal approximators, *Neural Networks*, 2(5), 359, 1989.

Jacobs, R. A., Increased rates of convergence through learning rate adaptation, *Neural Networks*, 1, 295, 1988.

Jin, L., Nikiforuk, P. N., and Gupta, M. M., Approximation capability of feedforward and recurrent neural networks, in Gupta, M. M., and Sinha, N. K., Eds., *Intelligent Control Systems: Concepts and Applications*, IEEE Press, 235, 1995.

Johansson, E. M., Dowla, F. U., and Goodman, D. M., Backpropagation learning for multilayred feed-foward neural networks using the conjugate gradient method. *International Journal of Neural Systems***,** 2(4): 291, 1992.

Kim, Y. H., Lewis, F. L., and Abdallah, C. T., A dynamic recurrent neural-network-based adaptive observer for a class of nonlinear systems, *Automatica*, 33(8), 1539, 1997.

Kolen, J. F., *Exploring the Computational Capabilities of Recurrent Neural Networks*, Ph.D. Thesis, The Ohio State University, Columbus, 1994.

Levin, A. V. and Narendra, K. S., Control of nonlinear dynamical systems using neural networks – controllability and stabilization, *IEEE Transactions on Neural Networks*, 4(2), 192, 1993.

Levin, A. V. and Narendra, K. S., Control of nonlinear dynamical systems using neural networks – Part II: observability, identification, and control, *IEEE Transactions on Neural Networks*, 7(1), 30, 1996.

Li, J. H., Michel, A. N., and Porod, W., Qualitative analysis and synthesis of a class of neural networks, *IEEE Transactions on Circuits and Systems*, 35(8), 976, 1988.

Luenberger, D. G., *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Reading, MA, 1989.

Moller, M. F., A scaled conjugate gradient algorithm for fast supervised learning, *Neural Networks*, 6(4), 525, 1993.

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

Nerrand, O., Roussel-Ragot, P., Personnaz, L., and Dreyfus, G., Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms. *Neural Computation*, 5(2), 165, 1993.

Oppenheim, A. V. and Schafer, R. W., *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1999.

Ott, E., *Chaos in Dynamical Systems*. Cambridge University Press, London, 1993.

Pearlmutter, B. A., Fast exact multiplication by the Hessian. *Neural Computation*, 6(1), 147, 1994.

Pearlmutter, B. A., Gradient calculations for dynamic recurrent neural networks: a survey, *IEEE Transactions on Neural Networks*, 6(5), 1212, 1995.

Poggio, T. and Girosi, F., Networks for approximation and learning, *Proceedings of the IEEE*, 78(9), 1481, 1990.

Shewchuk, J. R., An introduction to the conjugate gradient method without the agonizing pain, School of Computer Science, Carnegie Mellon University, Pittsburgh, August 4, 1994.

Shink, J. J., Adaptive IIR filtering, *IEEE ASSP Magazine*, 4, 21, 1989.

Siegelmann, H. T. and Sontag, E. D., Turing computability with neural nets, *Applied Mathematics Letters*, 4, 77, 1991.

Siegelmann, H. T., Horne, B. G., and Giles, C. L., Computational capabilities of recurrent NARX neural networks, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 27, 208, 1997.

Touati-Ahmed, D. and Storey, C., Efficient hybrid conjugate gradient techniques, *Journal of Optimization Theory and Applications* 64(2), 379, 1990.

Tsoi, A. C. and Back, A. D., Discrete time recurrent neural network architectures: a unifying review, *Neurocomputing*, 15, 183, 1997.

Tsoi, A. C. and Back, A. D., Locally recurrent globally feedforward networks: a critical review of architectures, *IEEE Transactions on Neural Networks*, 5(2): 229, 1994.

van der Smagt, P. P., Minimisation methods for training feedforward neural networks, *Neural Networks*, 7(1), 1, 1994.

Von Zuben, F. J. and Netto, M. L. A, Second-order training for recurrent neural networks without teacher-forcing, *Proceedings of the IEEE International Conference on Neural Networks*, 2, 801, 1995.

Von Zuben, F. J. and Netto, M. L. A., Recurrent neural networks for chaotic time series prediction, in Balthazar, J.M., Mook, D.T., and Rosário, J.M., Eds., *Nonlinear Dynamics, Chaos, Control, and Their Applications to Engineering Sciences*, 1, 347, 1997.

Williams, R. J., Adaptive state representation and estimation using recurrent connectionist networks, in Miller, W.T., Sutton, R.S., and Werbos, P., Eds., *Neural Networks for Control*, MIT Press, Cambridge, 1990.

Williams, R. J. and Zipser, D., A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, 1, 270, 1989.