INTERPRETATION OF RECURRENT NEURAL NETWORKS

Morten With Pedersen and Jan Larsen
CONNECT, Department of Mathematical Modelling, Building 321
Technical University of Denmark, DK-2800 Lyngby, Denmark
Phones: + 45 4525 + ext. 3920,3923 Fax: + 45 45872599
emails: mwp@imm.dtu.dk, jl@imm.dtu.dk

Abstract - This paper addresses techniques for interpretation and characterization of trained recurrent nets for time series problems. In particular, we focus on assessment of effective memory and suggest an operational definition of memory. Further we discuss the evaluation of learning curves. Various numerical experiments on time series prediction problems are used to illustrate the potential of the suggested methods.

INTRODUCTION

It is widely recognized that recurrent neural networks (RNNs) are flexible tools for time series processing, system identification and control problems, see e.g., [3]. Feed-forward networks can accommodate dynamics by having a lag space of past input and target values; however, a fully recurrent network with internal feedbacks allows for even more sophisticated dynamics. While fully RNN architectures are the ultimate tool for modeling dynamic relations, the comprehension of the networks is a challenging subject of ongoing research. Theoretical investigations of modeling capabilities of RNNs have been reported, see e.g., [2], [4], [7]. However, to the authors knowledge, there is no general theory of the dynamic behavior of a general RNN except for very special models like the Hopfield network, see e.g., [3]. This indeed indicates that theoretical analysis of RNNs is extremely complicated. On the other hand, one might pursue a more computational approach. The general computational tools from non-linear dynamic systems analysis like phase portraits, stability analysis, measurement of fractal dimensions or Lyapunov exponents (see e.g., [1], [3]) may be applied to the analysis of RNNs.

The motivation for this paper is evaluation and interpretation of *trained* recurrent networks, and to suggest and discuss simple operational techniques. In particular, we focus on the *learning curve* and present a new method to determine the effective *memory* of a recurrent network which conveys the relevant time scale of the dynamics.

NETWORK ARCHITECTURE

The objective is to model a non-linear dynamic relation among a discrete-time input signal $x(t)$ and a discrete time target signal, $d(t)$. The general architecture of the RNN considered in this presentation is based on [5] and consists of a single hidden layer of fully connected nonlinear units and one output unit. In particular, we focus on a network with only one external input, viz. the most recent value, $x(t)$. That is, the only information available about previous inputs stems from the memory build up internally in the net. The advantage using these networks is that the tedious problem of determining the optimal lag space of previous inputs is converted into determining the optimal network architecture in terms of connections and number of hidden neurons.

The network has a linear output in order to allow for arbitrary dynamic range, and at time t the prediction of the target $d(t)$ is given by,

$$y(t) = \sum_{i=1}^{N_h} w_{oi} \cdot s_i(t) + w_{ob} \tag{1}$$

where N_h is the number of hidden units, w_{oi} is the weight to the output unit from hidden unit i and w_{ob} is the output bias weight. The ith *state*, $s_i(t)$, is the output of a hidden unit computed as

$$s_i(t) = f\left(\sum_{j=1}^{N_h} w_{ij} \cdot s_j(t-1) + w_{ix} \cdot x(t) + w_{ib}\right) \tag{2}$$

where w_{ij} is the weight to hidden unit i from hidden unit j, w_{ix} is the weight from the external input $x(t)$, and w_{ib} is the bias weight. $f(\cdot)$ is the nonlinear activation function $\tanh(x)$. Note that the update of the units is *layered* [5]: at each time step the hidden units are updated before the output unit.

TRAINING AND GENERALIZATION

Suppose we have a training set of related values of inputs and targets $\mathcal{T} = \{x(t), d(t)\}_{t=1}^{T}$ where T is the number of training samples. Training is done by adjusting the weights so as to minimize a cost function. Here we employ the sum of squared errors augmented by a simple weight decay regularization term

$$C(\mathbf{w}) = \frac{1}{2}\sum_{t=1}^{T} e^2(t) + \frac{\alpha}{2}|\mathbf{w}|^2 \quad , \quad e(t) = d(t) - y(t) \tag{3}$$

where $\mathbf{w}$ is the concatenated set of weights and α is a small regularization parameter. Training aims at minimizing the cost function $C(\mathbf{w})$ and is thoroughly treated for RNNs in [6].

Suppose that training provides the estimated weight vector $\widehat{\mathbf{w}}$. Let $\boldsymbol{\pi}$ be an initial state vector[1] of the "true" data generating system leading to the training set $\mathcal{T}$ and define an associated probability distribution[2] $p(\boldsymbol{\pi})$. Further, define $\mathbf{x}(t) = [x(t), x(t-1), \cdots x(T+1)]^\top$ and let $p\left(d(t), \mathbf{x}(t) \,|\, \mathcal{T}, \boldsymbol{\pi}\right),\ t > T$, be the true joint probability density function of $[d(t), \mathbf{x}(t)]$ conditioned on the initial state $\boldsymbol{\pi}$ and the training set $\mathcal{T}$. The true joint p.d.f. is assumed to be time-independent (i.e., stationary). The generalization error of the trained net is defined as the expected squared prediction error on future data *immediately* succeeding the training data, i.e., for $t > T$,

$$G(\widehat{\mathbf{w}}) = \int [d(t) - y(t; \widehat{\mathbf{w}})]^2 \cdot p\left(d(t), \mathbf{x}(t) \,|\, \mathcal{T}, \boldsymbol{\pi}\right) \cdot p(\boldsymbol{\pi})\, dd(t) d\mathbf{x}(t) d\boldsymbol{\pi} \qquad (4)$$

Thus the generalization error is the ensemble average of the squared error over 1) possible realizations of $[d(t), \mathbf{x}(t)]$ due to inherent stochastic processes in the data generating system, and 2) over possible initial states leading to the particular training set.

We estimate the generalization error by,

$$\widehat{G}(\widehat{\mathbf{w}}) = \frac{1}{V} \sum_{t=T+1}^{T+V} e^2(t; \widehat{\mathbf{w}}) \qquad (5)$$

where V is the number of test samples.

LEARNING CURVE

The learning curve expresses the average generalization error over all possible training sets of a particular size T as a function of T and is an important tool for verifying whether enough data is available for proper training of the network. Moreover, the shape of the curve provides insight into the nature of the problem as demonstrated in the experimental section.

Practical considerations may lead to more restricted definitions. Here we compute the learning curve as the estimated generalization error when gradually expanding the training set. That is, there is no average over different sets of a particular size.

NETWORK MEMORY

A characteristic of recurrent neural networks is their ability to build up an internal memory representing the "history" of previous inputs on which the predictions of future values is based. The significance of this internal memory is especially clear when using RNNs having only one external input. Without the ability to create internal memory this class of networks would be useless.

Once a recurrent network is trained, the basic idea here is to define an integer variable M which expresses the effective memory of past values of

[1] The initial state captures the all information about the time series for $t \leq 0$.

[2] E.g., that all initial states are equally likely.

the input signal $x(t)$. The memory thus provides a partial insight into the functionality and dynamics of the network. The experimental section gives examples of interpreting the dynamics using this simple concept. Recurrent networks with only one external input can not give individual contribution to each previous input $x(t-m)$ but must store their own representation. Consequently, the RNN has a certain *memory profile*. We are currently pursuing the idea of determining the memory profile.

A feed-forward network does not possess any internal memory, i.e., the memory is explicitly determined by the memory contained in the preprocessing of the input signal. The standard approach is to feed the signals from a tapped delay line $[x(t), x(t-1), \cdots x(t-M)]$ into the network and the memory thus equals M.

The capacity of the internal memory of a recurrent network increases when the number of hidden units (i.e., the dimension of the state vector) increases as the state vector contains all information about previous inputs. However, to our knowledge, there is no reports on quantizing the notion of memory in recurrent networks. In the following we attempt to provide a definition of the memory of a specific trained recurrent network.

The output from the RNN defined in (1), (2) is based on the current and – in principle – infinitely many previous inputs[3], as shown by,

$$y(t) = y\left(t|\widehat{\mathbf{w}}, x(t), x(t-1), \ldots, x(-\infty)\right). \tag{6}$$

In order to determine the effective *average* memory of the recurrent network we suggest to evaluate an estimate of the generalization error, i.e., prediction error on a test set, using predictions based on only a *limited* number of previous inputs. This generalization error is then compared to the error obtained using all – in principle infinitely many – previous inputs.

In particular, when evaluating the generalization error using only the m most recent inputs, we compute,

$$\widehat{G}_m(\widehat{\mathbf{w}}) = \frac{1}{V}\sum_{t=T+1}^{T+V}\left[d(t) - y\left(t|\widehat{\mathbf{w}}, x(t), x(t-1), \ldots, x(t-m)\right)\right]^2, \; m \geq 0 \tag{7}$$

where V is the size of the test set. $y\left(t|\widehat{\mathbf{w}}, x(t), x(t-1), \ldots, x(t-m)\right)$ is computed for each $t \in [T+1; T+V]$ by *resetting*[4] the states $s_i(t-m-1)$, $i = 1, 2, \cdots, N_h$, to zero and then iterate the network from time $t-m$ until time t, using the output $y(t)$ at this time as the prediction of $d(t)$. In the first iteration, calculating $y\left(t-m|\widehat{\mathbf{w}}, x(t-m)\right)$, the network thus functions as a feed-forward network since the previous states of the hidden units – and thereby all previous external inputs – have no influence on the network output. Then, the network gradually builds up a representation of the past in

[3]This is also true for a RNN in which previous values of the output is fed back to the input.

[4]Setting the hidden unit states $s_i(t-m-1)$ to zero is equivalent to erasing the memory of the network regarding inputs before time $t-m$.

the hidden units during the next $m+1$ iterations before it makes its prediction at time t.

The resulting errors $\widehat{G}_0(\widehat{\mathbf{w}}), \widehat{G}_1(\widehat{\mathbf{w}}), \ldots$ are then compared to $\widehat{G}_\infty(\widehat{\mathbf{w}})$ denoting the error obtained when using all available previous inputs, i.e., no resetting of the hidden unit states at any time. The memory M is now defined as,

$$M = \inf \left\{ m \;\middle|\; \forall\, m' \geq m,\; \frac{|\widehat{G}_{m'}(\widehat{\mathbf{w}}) - \widehat{G}_\infty(\widehat{\mathbf{w}})|}{\widehat{G}_\infty(\widehat{\mathbf{w}})} < \epsilon \right\} \tag{8}$$

where ϵ is a *small* number. Thus, the memory, M, denotes the minimal number of previous inputs beyond which additional inputs are insignificant.

The memory measure outlined above determines the number of previous inputs that the network needs knowledge about in order to obtain good predictions on *all* samples in the test set. Thus the measure can be interpreted as the *average* memory of the network. A recurrent network, however, is a *dynamic* system whose internal characteristics can be highly influenced by the nature of the input series. Especially, if the input series exhibits regions of non-stationary behavior, the network dynamics including memory must clearly be affected. Such changes in dynamics are not captured by the average memory measure and we may define a *local memory*, in accordance with (8), using a local generalization error estimate[5]

$$\widehat{G}_m(t; \widehat{\mathbf{w}}) = \frac{1}{K} \sum_{t'=t-K+1}^{t} [d(t') - y\,(t'|\widehat{\mathbf{w}}, x(t'), x(t'-1), \ldots, x(t'-m))]^2, \tag{9}$$

where $m \geq 0$, $t > T$, and $1 \leq K \leq V$ is the size of a smaller test set. Choosing K too small gives rise to a very noisy measure of the generalization error; however, in principle a good resolution of changes in memory requirement. On the other hand, increasing K improves generalization accuracy but reduces the resolution of changes in memory.

EXPERIMENTS

The proposed methods for estimating the learning curves and memory are evaluated on two chaotic time series prediction problems, viz. the laser series from the Santa Fe time series competition [9] and the artificially generated Mackey-Glass series [8].

The laser series is illustrated in the left panel of Figure 1. Let $z(t)$ denotes the series, then identification is done by training the network to perform a one step ahead prediction, i.e., we use $x(t) = z(t)$ and $d(t) = z(t+1)$. All available 10093 samples are used and scaled to zero mean and unit variance. From these data we construct a learning curve. The training series are obtained by

[5]Notice, by defining this measure for all $t > T$ some of the first values are based partly on training examples.

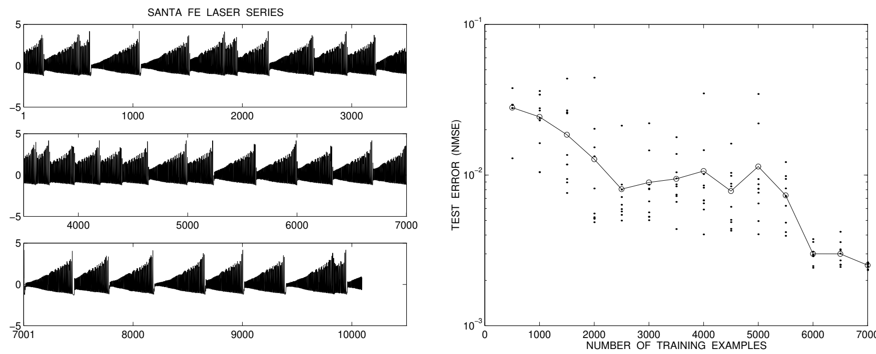

Figure 1: Left panel: The Santa Fe laser series. Right panel: Learning curve for the laser data. Dots denote error for individual nets, the connected circles indicate the average.

extending backwards in time from point 7000 and the last 3093 points in the series are used as test series. For instance, a training set of size 1000 involves training using $z(6000)$ through $z(7000)$. The employed nets have one external input and ten hidden units. For each number of increasing training set sizes, we train ten networks using different random initial weights and compute the resulting normalized mean squared error (NMSE) on the test set. NMSE is defined by

$$\mathrm{NMSE} = \frac{|\mathcal{S}|^{-1} \sum_{t \in \mathcal{S}} e^2(t; \widehat{\mathbf{w}})}{\widehat{\mathrm{var}}(d(t))} \tag{10}$$

where t runs over the set $\mathcal{S}$ in question (i.e., either training or test set), $|\mathcal{S}|$ is the size of the set, and $\widehat{\mathrm{var}}(\cdot)$ denotes the empirical variance.

The learning curve is shown in the right panel of Figure 1. Initially the test error drops as the size of the training set is increased, but from training set size 2500 to 5500 the average test error is fairly constant. This can be explained by visual inspection of the laser series as the "shape" of many collapses between the corresponding points 1500–4500 seems atypical for the test series. We see a significant drop in test error when increasing the training set size from 5500 to 6000 points which might be explained by the fact that the training set now incorporates an additional collapse very similar in shape to the ones in the test series. These observations suggest that for the laser series, the concept of an example should be conceived on several time scales: there are the pointwise examples corresponding to each single input presented to the network; but more important, there obviously exists "super examples" consisting of a whole section of the time series. If additional super examples or sections are not similar to the sections encountered in the test series, generalization will not improve as seen in the right panel of Figure 1.

We now examine the memory of selected networks. The left panel of Figure 2 depicts the normalized version of Eq. (7) for increasing values of lag space m when evaluating one of the networks with low test error trained on 7000 examples. The horizontal dotted line indicates the normalized level

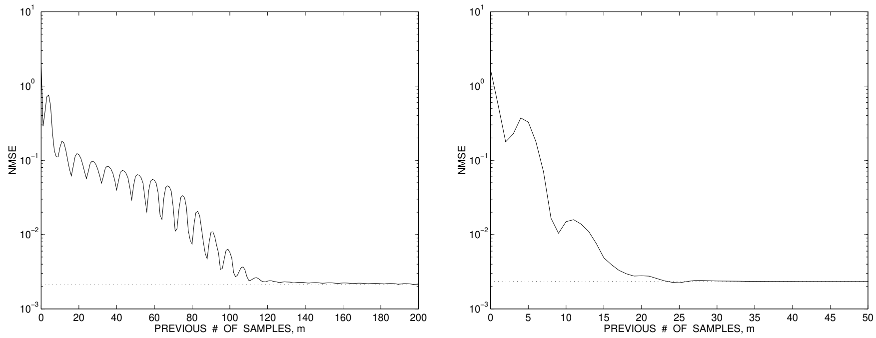

Figure 2: Left panel: Measuring *average* memory for one of the networks with low generalization error trained on 7000 examples from the laser series. Right panel: Measuring *average* memory for another of the networks trained on 7000 examples.

$\widehat{G}_\infty(\widehat{\mathbf{w}})$ using all available previous inputs. It seems that the network has a memory somewhere between 120 and 200. The precision ϵ in (8) denotes a level below which we consider the two errors as equivalent. The value of the memory thus naturally depends on the choice of ϵ as shown in Table 1. In the right panel of Figure 2 the normalized test error for increasing lag

ϵ	0.05	0.025	0.01
M	150	183	198

Table 1: The value of the memory dependence on ϵ for curve in the left panel of Figure 2.

space m for another of the nets trained on 7000 points is shown. We note that for this network the memory M is less sensitive to ϵ, as it is between 23–25 for $\epsilon \leq 0.18$. We also note that the memory is much shorter than for the previous network even though the test errors are almost identical. Note, since the network complexity[6] is restricted, a network with short memory is able to allow for more individual contribution of each of the previous inputs $x(t-n)$ than a network with long memory. The memory profile of a short term memory net is thus more fine grained than that of a long term memory net (with the same complexity). One might claim that a compact memory model is better tuned to the problem.

In the left panel of Figure 3 we illustrate the average memory of the network with lowest test error when training on only 500 examples. We notice that by limiting the memory the error can actually become lower than $\widehat{G}_\infty$. This effect often occurs for *overtrained* networks which is also the case here. The memory of the network is highly specialized on the training set; limiting the memory acts as regularization and actually improves the performance on the test set.

We now illustrate that the memory of a recurrent network indeed is a

[6]E.g., measured by the number of hidden neurons.

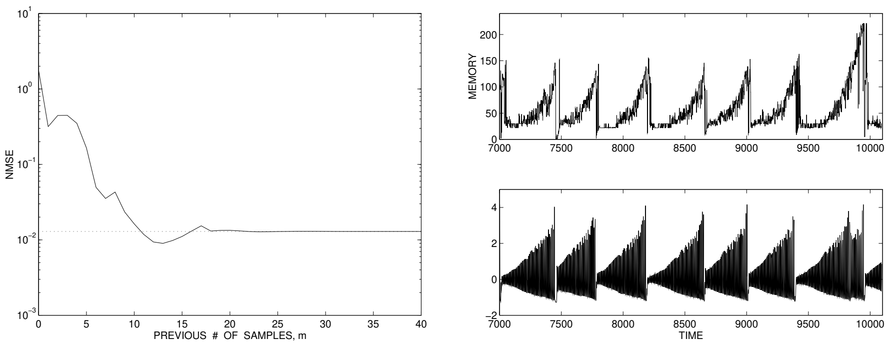

Figure 3: Left panel: Measuring *average* memory for best network trained on 500 examples from the laser series. Right panel: Measuring *local* memory with threshold $\epsilon = 0.01$ using five point average, $K = 5$.

dynamic quantity by examining the *local* memory defined by Eq. (8) and (9) for the network whose *average* memory is shown in the left panel of Figure 2. The right panel of Figure 3 and the left panel of Figure 4 illustrate the dynamic memory measure using precision $\epsilon = 0.01$ and averaging over $K = 5$ and $K = 50$ examples, respectively. The memory is seen to be very dependent upon where in the laser series it is measured; the closer to a collapse, the larger. The memory required around the last collapse is significantly larger than around the previous collapses. This may be explained by the observation that the characteristics of the laser series just before the last collapse is highly atypical from the rest of the test series. The memory in the right panel of Figure 3 averaging over only $K = 5$ previous errors is seen to be a very noisy quantity. As K is increased the error measure becomes smoother. Recall from Table 1 that the average memory for $\epsilon = 0.01$ is $M = 198$; however, the illustrations of the local memory shows that by omitting the last collapse the average memory would be measured to 150, approximately.

The Mackey-Glass series is a standard problem of nonlinear dynamics and results from the integration of a differential equation, see e.g., [8]. Standard

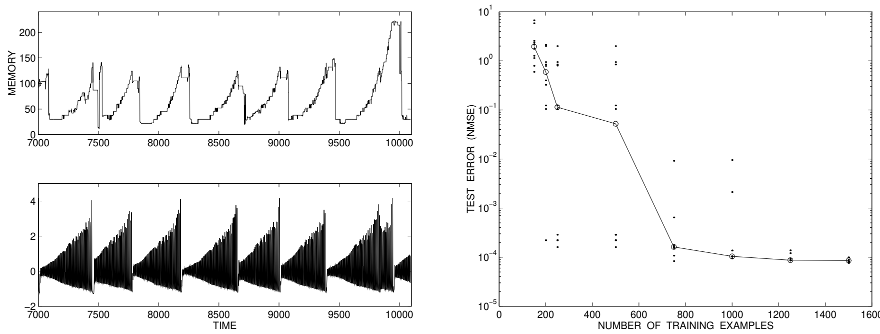

Figure 4: Left panel: Measuring *local* memory with threshold $\epsilon = 0.01$ using fifty point average, $K = 50$. Right panel: Learning curve for the Mackey-Glass series.

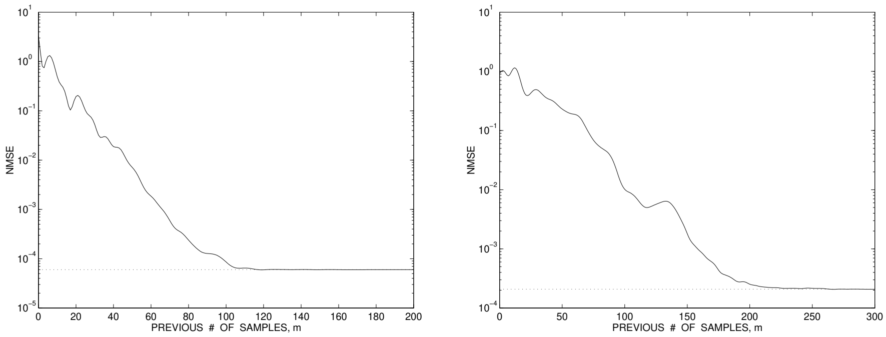

Figure 5: Measuring *average* memory for networks trained on 1500 examples from the Mackey-Glass series. Left Panel: Network having short memory. Right panel: Network having long memory.

practice is to implement a six step ahead predictor, i.e., modeling $z(t)$ from a lag space vector $\mathbf{x}(t) = [z(t-6), z(t-12), \ldots, z(t-6n_I)]$ using feed-forward networks. Here we implement the six step ahead predictor with target value $d(t) = z(t)$ using a recurrent network with only one external input, $x(t) = z(t-6)$, and ten hidden units. In the right panel of Figure 4 is shown a learning curve for the Mackey-Glass series when training on up to 1500 samples and testing on the following 7000 samples. For each training set size ten networks were trained. The learning curve indicates that more than 1000 examples are needed in order to obtain consistently good results on the test set. We then determined the average memory defined by Eq. (7) for the properly trained networks with the lowest errors on the test set. Using the threshold $\epsilon = 0.01$ we found that the networks implemented a memory in the range of 118–263, as seen from Figure 5.

The memories implemented by the recurrent networks are surprisingly long. In order to obtain comparable performance using feed-forward networks six external inputs are needed, thus spanning a total of only 31 previous samples. This is the minimal memory neccessary for good performance provided weighting of individual lags is possible, however, a RNN's memory profile is more coarse grained reducing the possibilty of individual weighting. Furthermore, maintaining information about all previous input values seems to bias recurrent networks towards the implementation of a long *effective* memory.

The long memory implemented by the recurrent networks seems to be of prime importance for the *robustness* of these models. Preliminary experiments indicate that recurrent networks are far more resilient to noise pertubations of the input data than comparable feed-forward networks. Examination of the robustness of recurrent networks is a topic of ongoing research.

CONCLUSION

In this paper we have focused on determining the *effective* memory of recurrent neural networks when used for time series processing, equivalent to the span of the externally provided lag space for feed-forward networks. In particular, we have suggested an operational definition which measures the memory of a fully trained RNN on a test set. The viability of the method is illustrated on two chaotic time series problems.

ACKNOWLEDGMENTS

This research was supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT). JL furthermore acknowledge the Radio Parts Foundation for financial support. Lars Kai Hansen is acknowledged for stimulating discussions.

REFERENCES

[1] H.D.I. Abarbanel: **Analysis of Observed Chaotic Data**, New York, NY: Springer-Verlag, 1996.

[2] M. Casey: "The Dynamics of Discrete-Time Computation, with Application to Recurrent Neural Networks and Finite State Machine Extraction," **Neural Computation**, vol. 8, pp. 1135–1178, 1996.

[3] S. Haykin: **Neural Networks: A Comprehensive Foundation**, New York, New York: Macmillan College Publishing Company, 1994.

[4] T. Lin, B.G. Horne, P. Tino & C.L. Giles: "Learning Long-term Dependencies with NARX Recurrent Neural Networks," **IEEE Transactions on Neural Networks**, vol. 7, no. 6, p. 1329, 1996.

[5] M.W. Pedersen & L.K. Hansen: "Recurrent Networks: Second Order Properties and Pruning," in G. Tesauro, D. Touretzky & T. Leen (eds.) **Advances in Neural Information Processing Systems 7**, Cambridge, MA: The MIT Press, 1995, pp. 673–680.

[6] M.W. Pedersen: "Training Recurrent Networks," in **Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VII**, Piscataway, New Jersey: IEEE, 1997.

[7] H.T. Siegelmann, B.G. Horne & C.L. Giles: "Computational Capabilities of Recurrent NARX Neural Networks," **Technical Report UMIACS-TR-95-12, IEEE Transactions on Systems, Man and Cybernetics**, 1997 (in press).

[8] C. Svarer, L. K. Hansen, J. Larsen & C. E. Rasmussen: "Designer Networks for Time Series Processing," in C. A. Kamm, G. M. Kuhn, B. Yoon, R. Chellappa & S. Y. Kung (eds.), **Proceedings of the IEEE Workshop on Neural Networks for Signal Processing 3**, Piscataway, New Jersey: IEEE, pp. 78–87, 1993.

[9] A.S. Weigend, & N.A. Gershenfeld (eds.): **Time Series Prediction: Forecasting the Future and Understanding the Past**, Santa Fe Institute Studies in the Sciences of Complexity, Reading, MA: Addison-Wesley, 1993.