In *Time Series Prediction. Forecasting the Future and Understanding the Past*, Eds. by A. Weigend and N. Gershenfeld, SFI Studies in the Sciences of Complexity, Proc. Vol XVII, Addison-Wesley, 1994.

Time Series Prediction by Using a Connectionist Network with Internal Delay Lines

Eric A. Wan
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-4055
e-mail: wan@isl.stanford.edu

Abstract

A neural network architecture, which models synapses as Finite Impulse Response (FIR) linear filters, is discussed for use in time series prediction. Analysis and methodology are detailed in the context of the Santa Fe Institute Time Series Prediction Competition. Results of the competition show that the FIR network performed remarkably well on a chaotic laser intensity time series.

1 Introduction

The goal of time series prediction or forecasting can be stated succinctly as follows: given a sequence $y(1)$, $y(2)$, $\ldots y(N)$ up to time N, find the continuation $y(N+1)$, $y(N+2)$... The series may arise from the sampling of a continuous time system, and be either stochastic or deterministic in origin. The standard prediction approach involves constructing an underlying model which gives rise to the observed sequence. In the oldest and most studied method, which dates back to Yule [1], a linear autoregression (AR) is fit to the data:

$$y(k) = \sum_{n=1}^{T} a(n)y(k-n) + e(k) = \hat{y}(k) + e(k). \tag{1}$$

This AR model forms $y(k)$ as a weighted sum of past values of the sequence. The single step prediction for $y(k)$ is given by $\hat{y}(k)$. The error term $e(k) = y(k) - \hat{y}(k)$ is often assumed to be a white noise process for analysis in a stochastic framework.

More modern techniques employ *nonlinear* prediction schemes. In this paper, neural networks are used to extend the linear model. The basic form $y(k) = \hat{y}(k) + e(k)$ is retained; however, the estimate $\hat{y}(k)$ is taken as the output $\mathcal{N}$ of a neural network driven by past values of the sequence. This is written as:

$$y(k) = \hat{y}(k) + e(k) = \mathcal{N}[y(k-1), y(k-2), ...y(k-T)] + e(k). \tag{2}$$

Note this model is equally applicable for both scalar and vector sequences.

The use of this nonlinear autoregression can be motivated as follows. First, *Takens Theorem* [2, 3] implies that for a wide class of deterministic systems, there exists a *diffeomorphism* (one-to-one differential mapping) between a finite window of the time series $[y(k-1), y(k-2), ...y(k-T)]$ and the underlying *state* of the dynamics system which gives rise to the time series. This implies that there exists, in theory, a nonlinear autoregression of the form $y(k) = g[y(k-1), y(k-2), ...y(k-T)]$, which models the series exactly (assuming no noise). The neural network thus forms an approximation to the ideal function $g(\cdot)$. Furthermore, it has been shown [4, 5, 6] that a feedforward neural network $\mathcal{N}$ with an arbitrary number of neurons is capable of approximating any *uniformly* continuous function. These arguments provide the basic motivation for the use of neural networks in time series prediction.

The use of neural networks for time series prediction is not new. Previous work includes that of Werbos[7, 8], Lapedes [9], and Weigend et al [10] to cite just a few. The connectionist entries in the *SFI Competition* attest to the success and significance of networks in the field. In this paper, we focus on a method for achieving the nonlinear autoregression by use of a Finite Impulse Response (FIR) network [11, 12]. We start by reviewing the FIR network structure and presenting its adaptation algorithm called *temporal backpropagation*. We then discuss the use of the network in a prediction configuration. The results of the *SFI Competition* are then presented with step by step explanations on how the specific predictions were accomplished. We conclude by re-evaluating our original prediction model of Eq. 2 and propose various classes of network schemes for both autoregressive and state-space models.

2 FIR Network Model

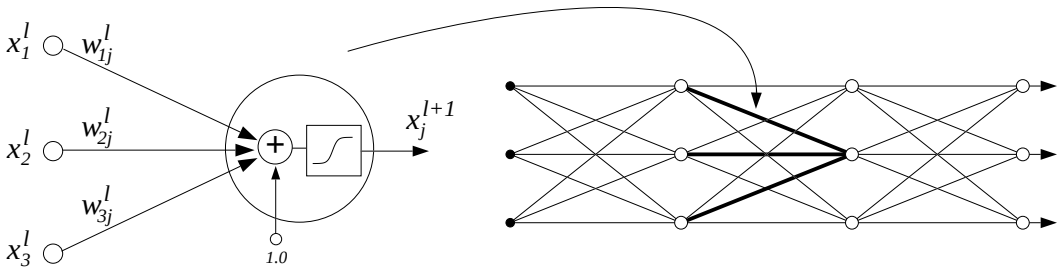

Figure 1: Static Neuron Model and Feedforward Network: each neuron passes the weighted sum of its inputs through a sigmoid function. The output of a neuron in a given layer acts as an input to neurons in the next layer. In the network illustration, each line represents a synaptic connection. No feedback connections exist.

The traditional model of the multilayer neural network is shown in Figure 1. It is composed of a layered arrangement of artificial neurons in which each neuron of a given layer feeds all neurons of the next layer. A single neuron extracted from the l^{th} layer of an L-layer network is also represented in the figure. The inputs x_i^l to the neuron are multiplied by

variable coefficients $w_{i,j}^l$ called *weights*, which represent the synaptic connectivity between neuron i in the previous layer and neuron j in layer l. The output of a neuron, x_j^{l+1}, is simplistically taken to be a sigmoid function[1] of the weighted sum of its inputs:

$$x_j^{l+1} = f\left(\sum_i w_{i,j}^l x_i^l\right). \tag{3}$$

A *bias* input to the neuron is achieved by fixing x_0^l to 1. The network structure is completely defined by taking x_i^0 to be the external inputs, and x_i^L to be the final outputs of the network. Training of the network can be accomplished using the familiar *backpropagation* algorithm [15].

The model of the feedforward network described above forms a complex mapping from the input of the first layer to the output of the last layer. Nevertheless, for a fixed set of weights, it is a *static* mapping; there are no internal dynamics. A modification of the basic neuron is accomplished by replacing each static synaptic weight by an FIR linear filter[2]. By FIR we mean that for an input excitation of finite duration, the output of the filter will also be of finite duration. The most basic FIR filter can be modeled with a tapped delay line as illustrated in Fig. 2. For this filter, the output $y(k)$ corresponds to a weighted sum of past delayed values of the input:

$$y(k) = \sum_{n=0}^{T} w(n)x(k-n). \tag{4}$$

Note that this corresponds to the *moving average* component of a simple Autoregressive Moving Average ($ARMA$) model [16, 17]. The FIR filter, in fact, was one of the first basic adaptive elements ever studied [21]. From a biological perspective, the synaptic filter represents a Markov model of signal transmission corresponding to the processes of axonal transport, synaptic modulation, and charge dissipation in the cell membrane [18, 19, 20].

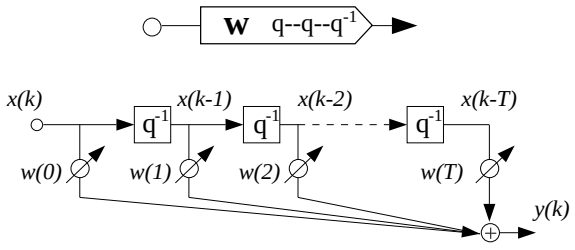

Figure 2: FIR Filter Model: A tapped delay line shows the functional model of the Finite Impulse Response "synapse" (q^{-1} represents a unit delay operator, i.e., $x(k-1) = q^{-1}x(k)$).

Continuing with notation, the coefficients for the synaptic filter connecting neuron i to neuron j in layer l is specified by the vector $\mathbf{w}_{ij}^l = \left[w_{i,j}^l(0), w_{i,j}^l(1), ..w_{i,j}^l(T^l)\right]$. Similarly

[1] The sigmoid is a continuous S shaped function chosen to roughly model the thresholding properties of a real neuron. Mathematically, it is common to use the hyperbolic tangent function, $tanh$.

[2] The Infinite Impulse Response (IIR) filter has the form $y(k) = \sum_{n=0}^{T} a(n)x(k-n) + \sum_{m=0}^{M} b(m)y(k-m)$ and will not be considered in this paper.

$\mathbf{x}_i^l(k) = \left[x_i^l(k), x_i^l(k-1), ..x_i^l(k-T^l)\right]$ denotes the vector of delayed states along the synaptic filter. This allows us to express the operation of a filter by a vector dot product $\mathbf{w}_{i,j}^l \cdot \mathbf{x}_i^l(k)$, where time relations are now implicit in the notation. The output $x_j^{l+1}(k)$ of a neuron in layer l at time k is now taken as the sigmoid function of the sum of all filter outputs which feed the neuron (Fig. 3):

$$x_j^{l+1}(k) \quad = \quad f\left(\sum_i \mathbf{w}_{i,j}^l \cdot \mathbf{x}_i^l(k)\right). \tag{5}$$

Note the striking similarities in appearance between these equations and those of the static model (Eq. 3) along with their associated figures. Notationally, scalars are replaced by vectors and multiplications by vector products. The convolution operation of the synapse is implicit in the definition. As we will see, these simple analogies carry through when comparing standard backpropagation for static networks to *temporal backpropagation* for FIR networks.

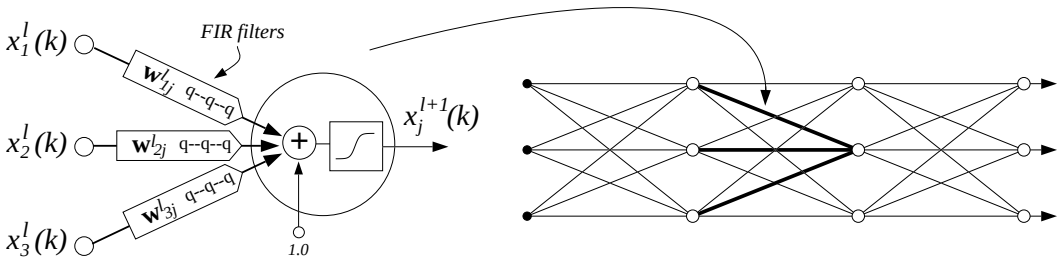

Figure 3: FIR Neuron and Network: In the temporal model of the neuron, input signals are passed through synaptic filters. The sum of the filtered inputs is passed through a sigmoid function to form the output of the neuron. In the feedforward network all connections are modeled as FIR filters.

2.1 Alternative Representations of the FIR Topology

A similar network structure incorporating embedded time delays is the *Time-Delay Neural Network* [13, 14, 22]. TDNNs have recently become popular for use in phoneme classification. A TDNN is typically described as a layered network in which the outputs of a layer are buffered several time steps and then fed fully connected to the next layer (see Fig. 4). The TDNN and the FIR network can, in fact, be shown to be functionally equivalent. Differences between the topologies are a matter of their pictorial representations and an added formalism in notation for the FIR network. In addition, the FIR network is more easily related to a standard multilayer network as a simple temporal or vector extension. As we will see, the FIR representation also leads to a more desirable adaptation scheme for on-line learning.

An alternative representation of the FIR network (and TDNN) can be found by using a technique referred to as *unfolding-in-time*. The general strategy is to remove all time delays by expanding the network into a larger equivalent static structure. As an example, consider the very simple network shown in Figure 5a. The network consists of three layers with a

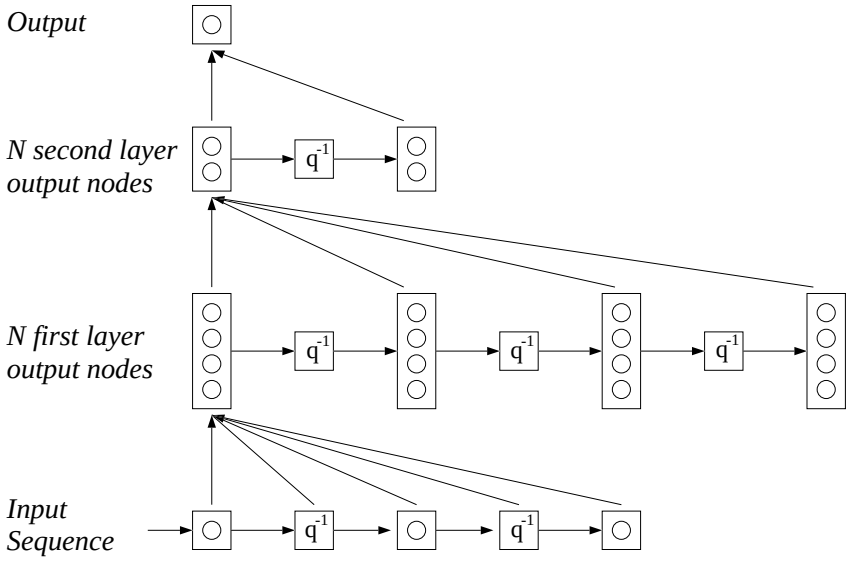

Figure 4: Time-Delay Neural Network: All node outputs in a given layer are buffered over several time steps. The outputs and the buffered states are then fed fully connected to the next layer. This structure is *functionally* equivalent to an FIR network.

single output neuron and two neurons at each hidden layer. All connections are made by second order (two tap) synapses. Thus while there are only 10 synapses in the network, there are actually a total of 30 variable filter coefficients. Starting at the last layer, each tap delay is interpreted as a "virtual neuron" whose input is delayed the appropriate number of time steps. A tap delay is then "removed" by replicating the previous layers of the network and delaying the input to the network accordingly (Fig. 5b). The process in then continued backward through each layer until all delays have been removed. The final unfolded network is shown in Figure 5c.

This method produces an equivalent *constrained* static structure where the time dependencies have been made external to the network itself. Notice that whereas there were initially 30 filter coefficients, the equivalent unfolded structure now has 150 static synapses. This can be seen as a result of redundancies in the static weights. In fact, the size of the equivalent static network grows *geometrically* with the number of layers and tap delays (see Table 1). In light of this, one can view an FIR network as a compact representation of a larger static network with imposed symmetries. These symmetries force the network to subdivide the input pattern into local overlapping regions. Each region is identically processed with the results being successively combined through subsequent layers in the network. This is in contrast to a fully connected network which may attempt to analyze the scene all at once. Similar locally symmetric constraints have been motivated for use in pattern classification using "shared weight" networks [23, 24].

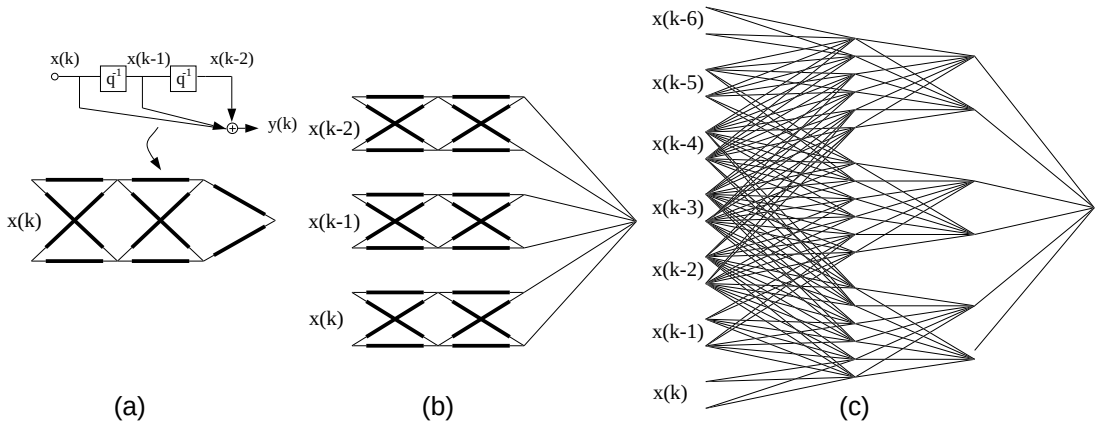

Figure 5: An FIR network with 2nd order taps for all connections is unfolded into a *constrained* static network. The original structure has 30 variable filter coefficients while the resulting network has 150 static synapses.

Table 1: FIR Network vs. Static Equivalent

Network Dimension		*Variable Parameters*	*Static Equivalent*
$Nodes^{\dagger}$	$Order^{\ddagger}$		
2×2×2×1	2:2:2	30	150
5×5×5×5	10:10:10	605	36,355
3×3×3	9:9	180	990
3×3×3×3	9:9:9	270	9,990
3^n	9^{n-1}	$(n-1)90$	$10^n - 10$

† Number of Inputs × Hidden Neurons × Outputs. ‡ Order of FIR synapses in each layer.

The relative size of the FIR network to the equivalent static structure is compared. As can be seen, the static network grows geometrically in size with the FIR time network.

3 Adaptation: Temporal Backpropagation

Given an input sequence $x(k)$, the network produces the output sequence $y(k) = \mathcal{N}[W, x(k)]$, where W represents the set of all filter coefficients in the network. For now, assume that at each instant in time a desired output $d(k)$ is provided to the network (we will reformulate this in terms of time series prediction in the next section). Define the instantaneous error $e^2(k) = ||d(k) - \mathcal{N}[W, x(k)]||^2$ as the squared Euclidean distance between the network output and the desired output. The objective of training corresponds to minimizing over W the cost function:

$$C = \sum_{k=1}^{K} e^2(k), \tag{6}$$

where the sum is taken over all K points in the training sequence. Regularization terms (e.g. constraints on W) are not considered in this paper. The most straight forward method for minimizing C is stochastic gradient descent. Synaptic filters are updated at each increment of time according to:

$$\mathbf{w}_{ij}^{l}(k+1) = \mathbf{w}_{ij}^{l}(k) - \eta \frac{\partial e^2(k)}{\partial \mathbf{w}_{ij}^{l}(k)}, \tag{7}$$

where η controls the learning rate.

The most obvious way to obtain the gradient terms involves first unfolding the structure into its static equivalent, and then applying standard backpropagation. Unfortunately, this leads to an overall algorithm with very undesirable characteristics. Backpropagation applied to a static network finds the gradient terms associated with each weights in the network. Since the constrained network contains "duplicated" weights, individual gradient terms must later be carefully recombined to find the *total* gradient for each unique filter coefficient. Locally distributed processing is lost as global bookkeeping becomes necessary to keep track of all terms; no simple recurrent formula is possible. These drawbacks are identical for the TDNN structure.[3]

A more attractive algorithm can be derived if we approach the problem from a slightly different perspective. The gradient of the cost function with respect to a synaptic filter is expanded as follows:

$$\frac{\partial C}{\partial \mathbf{w}_{ij}^{l}} = \sum_{k} \frac{\partial C}{\partial s_j^{l+1}(k)} \cdot \frac{\partial s_j^{l+1}(k)}{\partial \mathbf{w}_{ij}^{l}}, \tag{8}$$

where $s_j^{l+1}(k) = \sum_i \mathbf{w}_{i,j}^{l} \cdot \mathbf{x}_i^{l}(k)$ is the input to neuron j prior to the sigmoid. We may interpret $\partial C / \partial s_j^{l+1}(k)$ as the change in the total squared error over all time, due to a change at the input of a neuron at a single instant in time. Note that we are not expressing the total gradient in the traditional way as the sum of instantaneous gradients. Using this new expansion, the following stochastic algorithm is formed:

$$\mathbf{w}_{ij}^{l}(k+1) = \mathbf{w}_{ij}^{l}(k) - \eta \frac{\partial C}{\partial s_j^{l+1}(k)} \cdot \frac{\partial s_j^{l+1}(k)}{\partial \mathbf{w}_{ij}^{l}}. \tag{9}$$

A complete derivation of the individual terms in this equation is provided in Appendix A. The final algorithm, called *temporal backpropagation* can be summarized as follows:

$$\mathbf{w}_{ij}^{l}(k+1) = \mathbf{w}_{ij}^{l}(k) - \eta \delta_j^{l+1}(k) \cdot \mathbf{x}_i^{l}(k) \tag{10}$$

$$\delta_j^{l}(k) = \begin{cases} -2e_j(k) f'(s_j^{L}(k)) & l = L \\ f'(s_j^{l}(k)) \cdot \sum_{m=1}^{N_{l+1}} \boldsymbol{\delta}_m^{l+1}(k) \cdot \mathbf{w}_{jm}^{l} & 1 \le l \le L-1, \end{cases}$$

[3]TDNN's are typically used for classification in a batch mode adaptation. Training consists of fully buffering the states until the entire pattern of interest is captured and then using backpropagation though multiple "time-shifted" versions of the network. This can be shown to be equivalent to using a similar unfolded network as above.

where $e_j(k)$ is the error at an output node, $f'()$ is the derivative of the sigmoid function, and $\boldsymbol{\delta}_m^l(k) \equiv \left[\delta_m^l(k)\ \delta_m^l(k+1)\ldots\delta_m^l(k+T^{l-1})\right]$ is a vector of propagated gradient terms. We immediately observe that these equations are seen as the vector generalization of the familiar backpropagation algorithm. In fact, by replacing the vectors $\mathbf{x}$, $\mathbf{w}$, and $\boldsymbol{\delta}$ by scalars, the above equations reduce to precisely the standard backpropagation algorithm for static networks. Differences in the *temporal* version are a matter of implicit time relations and filtering operations. To calculate $\delta_j^l(k)$ for a given neuron we *filter* the δ's from the next layer backwards through the FIR synapses for which the given neuron feeds (see Fig. 6). Thus δ's are formed not by simply taking weighted sums, but by backward filtering. For each new input and desired response vector, the forward filters are incremented one time step and the backward filters one time step. The weights are then adapted on-line at each time increment.

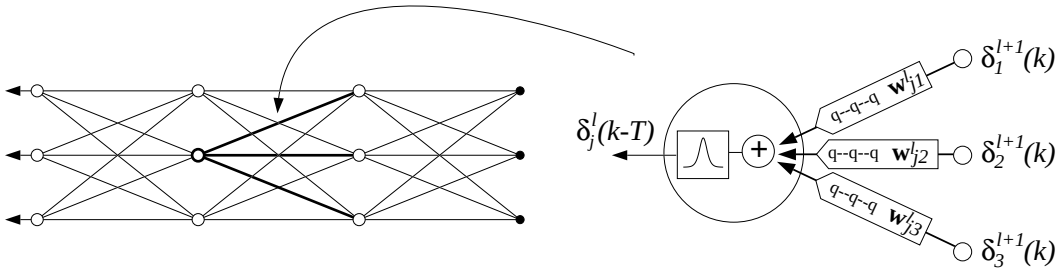

Figure 6: In temporal backpropagation, delta terms are *filtered* through synaptic connections to form the deltas for the previous layer. The process is applied layer by layer working backward through the network.

Temporal backpropagation preserves the symmetry between the forward propagation of states and the backward propagation of error terms. Parallel distributed processing is maintained. Furthermore, the number of operations per iteration now only grow *linearly* with the number of layers and synapses in the network. This savings comes as a consequence of the efficient recursive formulation. Each unique coefficient enters into the calculation only once in contrast to the redundant use of terms when applying standard backpropagation to the unfolded network[4].

4 Equation-Error Adaptation and Prediction

We are now in a position to discuss the use of the FIR network in the context of time series prediction. Recall that we wish to model the series $y(k)$ (for simplicity, we assume a scalar series). Figure 7a illustrates the basic predictor training configuration. At each time step,

[4]Temporal backpropagation may also be applied to TDNNs to form an efficient on-line algorithm in contrast to the typical batch mode training method.

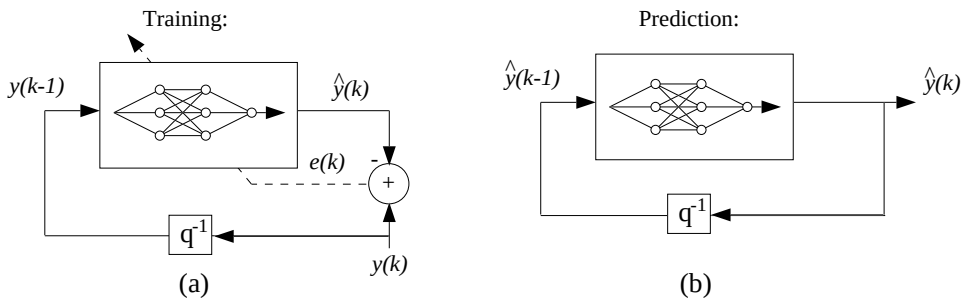

Figure 7: Network Prediction Configuration: The single step prediction $\hat{y}(k)$ is taken as the output of the network driven by previous sample of the sequence $y(k)$. During training the single step squared prediction error, $e^2(k) = (y(k) - \hat{y}(k))^2$, is minimized using temporal backpropagation. Feeding the estimate $\hat{y}(k)$ back forms a closed loop process used for long term iterated predictions.

the input to the FIR network is the known value $y(k-1)$, and the output $\hat{y}(k) = \mathcal{N}_q[y(k-1)]$ is the single step estimate of the true series value $y(k)$. Our model construct is thus:

$$y(k) = \mathcal{N}_q[y(k-1)] + e(k). \tag{11}$$

As explained earlier, the FIR network $\mathcal{N}_q$ may be represented as a constrained network acting on a finite window of the input (i.e. $\mathcal{N}_q[y(k-1)] \equiv \mathcal{N}_c[y(k-1), y(k-2)...y(k-T)]$). This allows us to rewrite Eq. 11 as:

$$y(k) = \mathcal{N}_c[y(k-1), y(k-2)...y(k-T)] + e(k), \tag{12}$$

which emphasizes the nonlinear autoregression.

During training, the squared error $e(k)^2 = (y(k) - \hat{y}(k))^2$ is minimized by using the temporal backpropagation algorithm to adapt the network ($y(k)$ acts as the desired response). Note we are performing *open-loop* adaptation; both the input and desired response are provided from the known training series. The actual output of the network is not fed back as input during training. Such a scheme is referred to as *equation-error* adaptation [25, 26]. The neural network community has more recently adopted the term *teacher-forcing* [27].

A simple argument for adapting in this fashion is as follows: in a stationary stochastic environment, minimizing the sum of the squared errors $e(k)^2$ corresponds to minimizing the *expectation* of the squared error:

$$E[e^2(k)] = E[y(k) - \hat{y}(k)]^2 = E[y(k) - \mathcal{N}_c[\mathbf{y}_1^T(k)]]^2, \tag{13}$$

where $\mathbf{y}_1^T(k) = [y(k-1), y(k-2)...y(k-T)]$ specifies the regressor.[5] The expectation is taken with respect to the joint distribution on $y(k)$ through $y(k-T)$. Now consider the identity:

$$E[y(k) - \mathcal{N}_c[\mathbf{y}_1^T(k)]]^2 = E[y(k) - E[y(k)|\mathbf{y}_1^T(k)]]^2 + E[\mathcal{N}_c - E[y(k)|\mathbf{y}_1^T(k)]]^2. \tag{14}$$

[5] In Eq. 13, $y(k)$ correspond to *random variables* in a stationary stochastic process, and not specific points in a given series.

The first term on the right hand side of this equation is independent of the estimator, and hence the optimal network map $\mathcal{N}_c^*$ is immediately seen to be,

$$\mathcal{N}_c^* = E[y(k)|\mathbf{y}_1^T(k)], \tag{15}$$

i.e. the conditional mean of $y(k)$ given $y(k-1)$ through $y(k-T)$, which is what we would have expected. Again we should emphasize that this only motivates the use of training a network predictor in this fashion. We cannot conclude that adaptation will necessarily achieve the optimum for a give structure and training sequence. Issues concerning *biased* estimators in the context of network learning are presented in [28].

Once the network is trained, long-term *iterated* prediction is achieved by taking the estimate $\hat{y}(k)$ and feeding it back as input to the network:

$$\hat{y}(k) = \mathcal{N}_q[\hat{y}(k-1)]. \tag{16}$$

This closed-loop system is illustrated in Fig. 7b. Equation 16 can be iterated forward in time to achieve predictions as far into the future as desired. Note, for a linear system, the roots of the regression coefficients must be monitored to insure that the closed loop system remains stable. For the neural network, however, the closed loop response will always have *bounded output* stability due to the sigmoids which limit the dynamic range of the network output.

Since training was based on only single step predictions, the accuracy of the long term iterated predictions can not be guaranteed in advance. One might even question the soundness of training open-loop when the final system is to be run closed-loop. In fact, *equation-error* adaptation for even a linear autoregression suffers from convergence to a *biased* closed-loop solution (i.e. $\theta = \theta^* +$ *bias*, where θ^* corresponds to the optimal set of closed loop autoregression parameters.) [26, 29]. An alternative configuration which adapts the closed loop system directly might seem more prudent. Such a set up is referred to as *output-error* adaptation. For the linear case, the method results in a estimator that is not biased. Paradoxically, however, the linear predictor may converge to a local minimum [30, 31, 32]. Furthermore, the adaptation algorithms themselves becomes more complicated and less reliable due to the feedback. As a consequence, we will not consider the *output-error* approach with neural networks in this paper.

4.1 Results of the SFI Competition

The plot in Figure 8 shows the chaotic intensity pulsations of an NH_3 laser [6] distributed as part of *The Santa Fe Institute Time Series Prediction and Analysis Competition*. For the laser data, only 1000 samples of the sequence were provided. The goal was to predict the next 100 samples. During the course of the competition, the physical background of the data set, as well as the 100 point continuation, was withheld to avoid biasing the final prediction results.

[6] "Measurements were made on an 81.5-micron 14NH3 cw (FIR) laser, pumped optically by the P(13) line of an N20 laser via the vibrational aQ(8,7) NH3 transition" - U. Huebner [33].

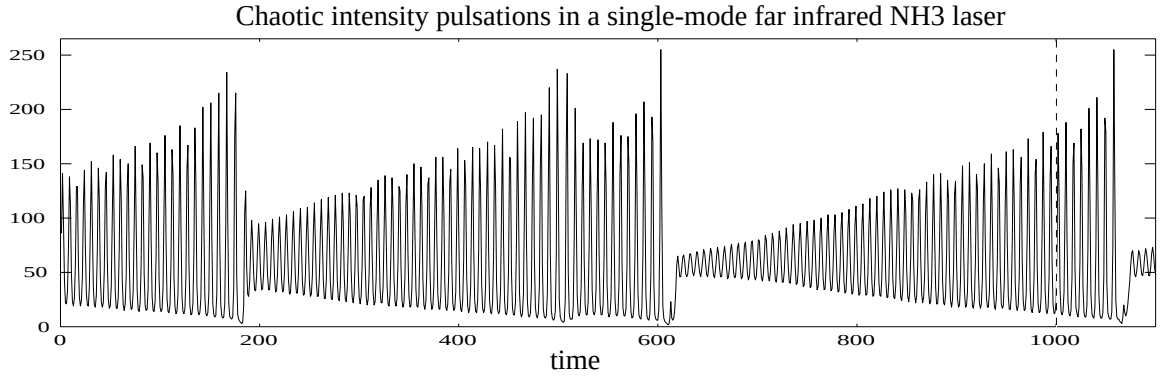

Figure 8: 1100 time points of chaotic laser data

Table 2: Normalized sum squared error prediction measures

Duration	*Single-Step Pred. nSSE*	*Iterated Pred. nSSE*
100-1000	0.00044	-
900-1000	0.00070	0.0026
1001-1050	0.00061	0.0061
1001-1100	0.02300	0.0551
1001-1100†	-	0.0273

† Prediction submitted for competition (75 iterations plus 25 smoothed values).

The 100 step prediction achieved using an FIR network is shown in Figure 9 along with the actual series continuation for comparison (the last 25 point of the prediction necessitated additional smoothing as will be explained under *Long Term Behavior*). It is important to emphasize that this prediction was made based on only the past 1000 samples. True values of the series for time past 1000 were not provided to the network nor were they even available when the predictions were submitted. As can be seen, the prediction is remarkably accurate with only a slight eventual phase degradation. A prediction based on a 25th order linear autoregression is also shown to emphasize the differences from traditional linear methods. Other submissions to the competition included methods of k-d trees, piecewise linear interpolation, low-pass embedding, SVD, nearest neighbors, Wiener filters, as well as standard recurrent and feedforward neural networks. The corresponding predictions may be found in this volume. As reported by this competition, the FIR network outperformed all other methods on this data set. While this is clearly just one example, these initial results are extremely encouraging. In the next sections we report on details concerning performance measures, selection of network parameters, training, testing, and postcompetition analysis.

Performance measure: A measure of fit is given by the *normalized sum squared error*:

$$nSSE = \frac{1}{\sigma^2 N} \sum_{k=1}^{N} (y(k) - \hat{y}(k))^2, \tag{17}$$

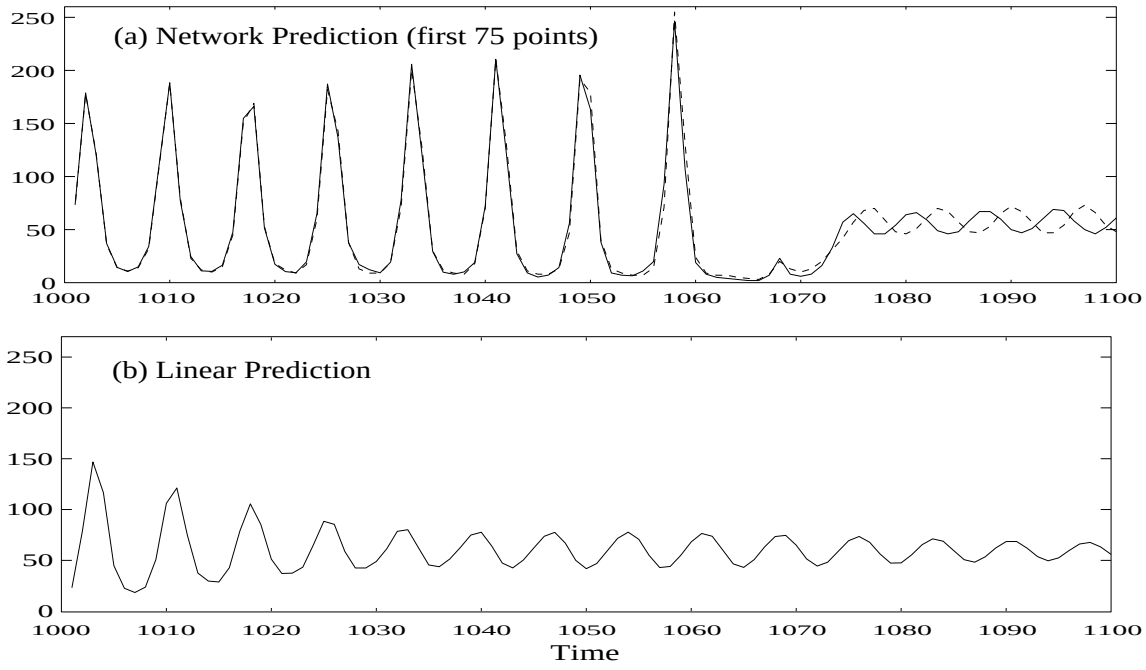

Figure 9: Time series predictions: (a) Iterated neural network prediction (first 75 points). The prediction is based only on the supplied 1000 points. Dashed line corresponds to actual series continuation. (b) 100 point iterated prediction based on a 25th order linear autoregression. Regression coefficients were solved using a standard least squares method.

where $y(k)$ is the true value of the sequence, $\hat{y}(k)$ is the prediction, and σ^2 is the variance of the true sequence over the prediction duration, N. A value of $nSSE = 1$ thus corresponds to predicting the unconditional mean. Table 4.1 summarizes various $nSSE$ values for both single-step and iterated predictions within the training set and for the continuation.

Selection of network dimensions: The FIR network used in the competition was a three layer network with 1x12x12x1 nodes and 25:5:5 taps per layer. Selection of these dimensions were based mostly on trial and error along with various heuristics. Since the first layer in the network acts as a bank of linear filters, selection of the filter order was motivated from linear techniques. As seen in Figure 10, the single step error residuals using linear AR predictors show negligible improvement for order greater than 15, while the autocorrelation indicates substantial correlation out to roughly a delay of 60. Candidate networks evaluated included 10, 50, and 100 order filters in the first layer with a varying number of units in the hidden layers. Attempts to perform analysis on the individual filters for the final converged network did not prove illuminating. In general, selection of dimensions for neural networks remains a difficult problem in need of further research.

Training and cross validation: For training purposes, the data was scaled to zero mean and variance 1.0. Initial weight values of the network were chosen randomly and then scaled to keep equal variance at each neuron output. The learning rate η was nominally set at 0.01 (this was selected heuristically and then varied during the course of training). Actual

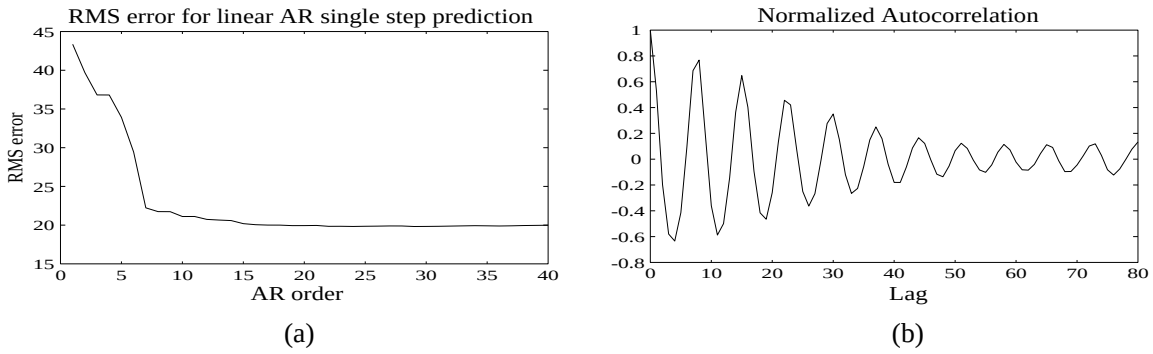

Figure 10: (a) Single step prediction errors using linear AR(N) filters. (b) Autocorrelation of laser data.

training occurred on the first 900 points of the series with the remaining 100 used for cross validation. A training run typically took over night on a Sun SPARC2 system and required several thousand passes through the training set (attempts to optimize training time were not actively pursued). Both the single step prediction error and the closed loop iterated prediction were monitored for the withheld 100 points. No overfitting was observed for the single step error; the larger the network the better the response. However, a larger network with a lower single step error often had a worse iterated prediction. Since the true task involved long term prediction, the iterated performance measure was ultimately used to evaluate the candidate networks. It was clear from the nature of the data that predicting downward intensity collapses would be the most important and difficult aspect of the series to learn. Since the withheld data contained no such features, iterated predictions starting near the point 550 were run to determine how well the network predicted the known collapse at point 600.

After the competition, the same network structure was retrained with different starting weights to access the sensitivity to initial conditions. Three out of four trials resulted in convergence to equivalent predictions.

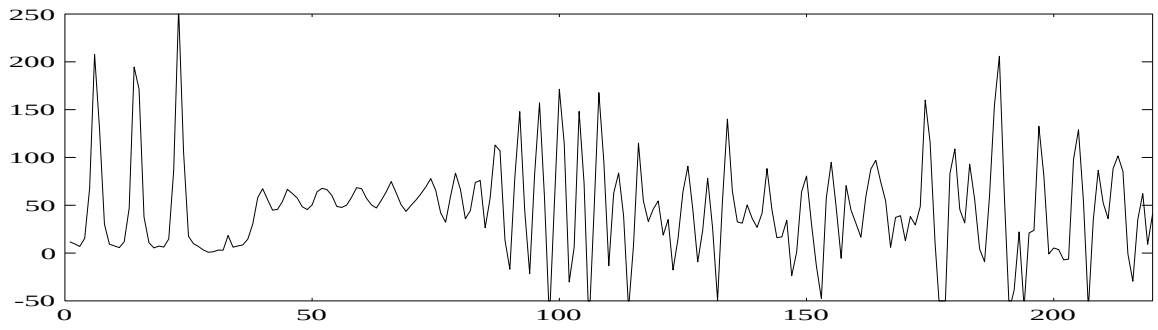

Figure 11: Extended iterated prediction.

Parameter fitting and long term behavior: The network used to learn the series had a total of 1105 variable parameter and was fit with only 1000 training points. (A statistician would

probably have a heart attack over this.) We speculate that this was necessary to accurately model the occurrence of a signal collapse from only two such examples in the training set. In a sense, we were forced to deal with learning what appears statistically to be an outlier. One must also realize that for this nonlinear system, the degrees of freedom do not correspond directly to the number of free parameters. We are really trying to fit a function which is constrained by the network topology. Issues concerning *effective* degrees of freedom are discussed in [38].

One consequence of the large number of parameters can be seen in the extended long term iterated prediction (Figure 11). Due to the excessive number of degrees of freedom, the signal eventually becomes corrupted and displays a noisy behavior. While a smaller network (or the addition of weight regularization) prevented this phenomena, such networks were unable to accurately predict the location of the intensity collapse. Note that the prediction still has *bounded output* stability due to the limiting sigmoids within the network.

Because of the eventual signal *corruption*, only the first 75 points of the actual iterated network prediction were submitted to the competition. This location corresponds to a few time steps after the detected intensity collapse and was chosen based on visual inspection of where the iterated prediction deteriorated. Since after an intensity collapse, the true series tends to display a simple slow growing oscillation, the remaining 25 points were selected by adjoining a similar sequence taken from the training set.

Error predictions: For the laser data, an estimate of the uncertainty of the prediction was also submitted. It was assumed that the true observed sequence, $y(k)$, was derived from an independent Gaussian process with a mean corresponding to the prediction $\hat{y}(k)$ and variance $\hat{\sigma}(k)^2$. Thus the standard deviations $\hat{\sigma}(k)$ determine *error bars* for each prediction value. A measure of the probability that the observed sequence was generated by the Gaussian model is given as the *negative average log-Likelihood*:

$$nalL = -\frac{1}{N}\sum_{k=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\hat{\sigma}(k)^2}}\int_{y(k)-0.5}^{y(k)+0.5} \exp -\frac{(\tau-\hat{y}(k))^2}{2\hat{\sigma}(k)^2} d\tau\right). \tag{18}$$

(See [34] for full explanation of likelihood measure). In order to estimate $\hat{\sigma}(k)^2$, we averaged the known iterated squared prediction errors starting at time count 400 through 550 (i.e. 150 separate iterated predictions were used). It clearly would have been more desirable to base these estimates on segments of data outside the actual training set; however, due to the limited data supplied, this was not possible. The error bars found using the above approach are shown with the prediction in Fig. 12, and correspond to $nalL = 1.51$. (The actual error bars submitted were scaled to much smaller values due to a missinterpretation of the performance measure, $nalL = 3.5$.) Alternative Bayesian methods for estimating uncertainties have been suggested by MacKay [35, 36] and Skilling [37].

Additional predictions: The complete 10000 point series continuation was provided after the competition. Fig. 13 shows various iterated predictions starting at different locations within the series. The original network (trained on only the first 1000 points) is used. While there is often noticeable performance degradation, the network is still surprisingly accurate at predicting the occurrence of intensity collapses.

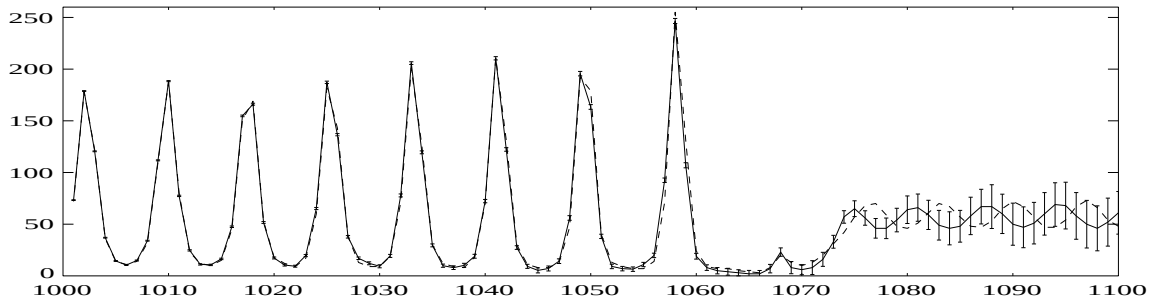

Figure 12: Estimated standard deviation error bars are centered at each prediction point. The dashed curve corresponds to the actual series.

Comments on other data sets in competition: The *SFI Competition* also distributed a financial series and 100,000 points of a high-dimensional computer generated chaotic series. Most of the effort, however, was made on the laser data. The financial series was not evaluated using an FIR network. Insufficient time and resources were the major stumbling blocks with the synthetic series. Cursory results with the FIR network were not encouraging; however, additional testing is necessary to draw solid conclusions.

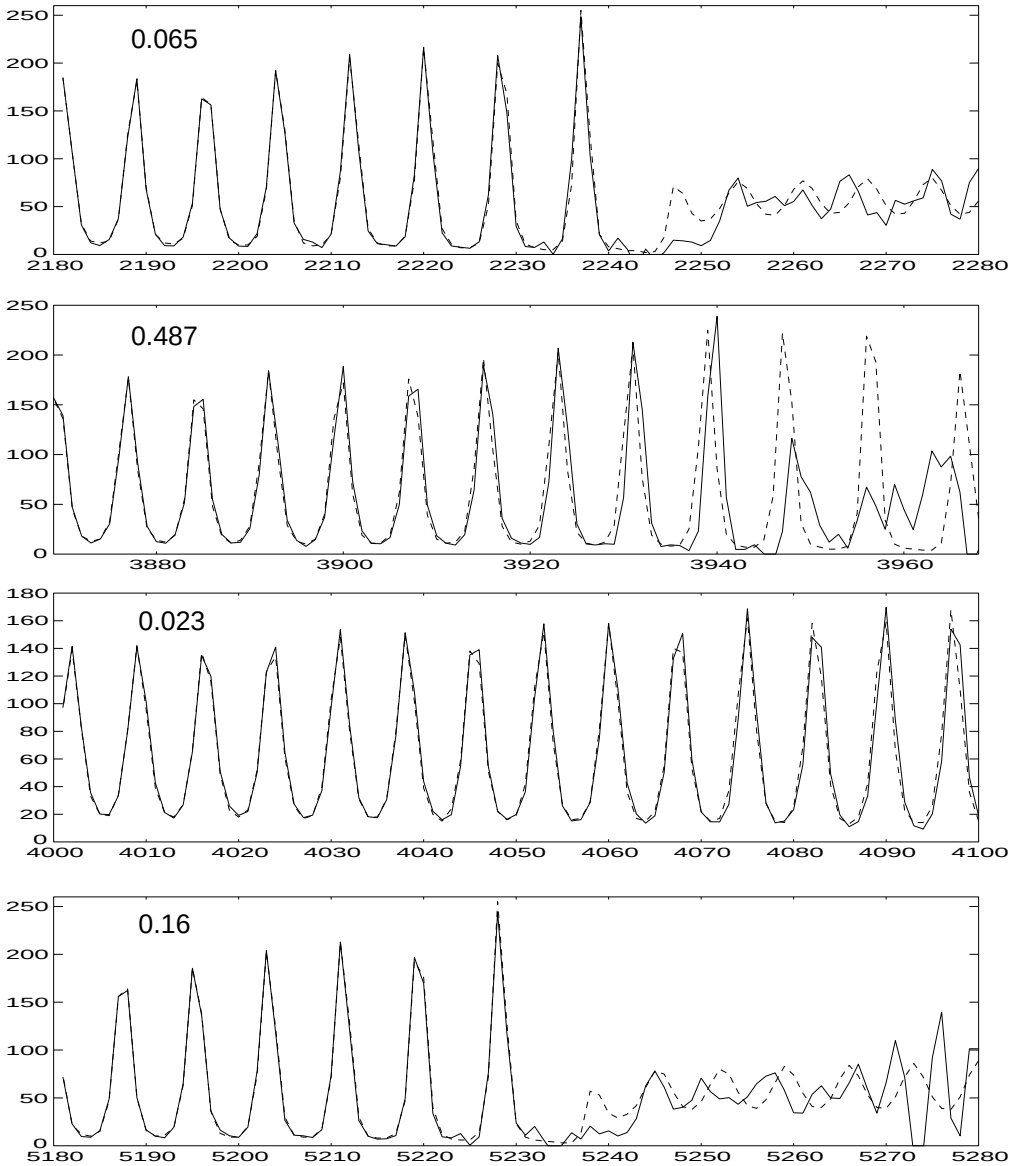

Figure 13: The original network, trained on the first 1000 points, is used to make iterated predictions at various starting locations. The dashed curves correspond to the true time series.

5 New Directions and Conclusions

In this paper we have focused on the basic autoregressive model; however, it should be clear that the general methodology presented may be easily extended to other configurations. The most trivial extension is to the nonlinear *Autoregressive Moving "Average"*:

$$y(k) = \mathcal{N}[y(k-1), y(k-2)...y(k-T), e(k-1), e(k-2)..e(k-T2)] + e(k), \tag{19}$$

where $\mathcal{N}$ may be an FIR network, a standard feedforward network, or any variety of network topologies. The *ARMA* model is most applicable for single-step prediction where we can additionally regress the network equations on past known prediction error residuals. Both the neural network *AR* and *ARMA* models, however, are extrapolated from rather old methods of linear difference equations. A more modern approach draws from *state-space* theory [39]. In the linear case the predictor corresponds to a Kalman Estimator [40]. Extending to neural networks yields the set of equations :

$$\mathbf{x}(k) = \mathcal{N}_1[\mathbf{x}(k-1), e(k-1)] \tag{20}$$

$$y(k) = \mathcal{N}_2[\mathbf{x}(k)] + e(k), \tag{21}$$

where $\mathbf{x}(k)$ corresponds to a vector of internal states which govern the dynamic system and must be learned by the network. This construct may form a more compact representation than that capable in an *ARMA* model and exhibit significantly different characteristics. Together, the *ARMA* and *state-space* models form a taxonomy of possible neural network prediction schemes. Such schemes must be investigated as the field of neural networks matures.

From this paper, we can conclude that an FIR network constitutes a powerful tool for use in time series prediction. The *SFI Competition* provided a forum in which the network was impartially benchmarked against a variety of other methods. While this was only one concrete example, we feel strongly that FIR networks, and neural networks in general, must be seriously considered when approaching any new time series problem.

Appendix A: Derivation of Temporal Backpropagation

Provided here is a complete derivation of the *temporal backpropagation* algorithm. We wish to minimize the cost function $C = \sum_k e^2(k)$ (i.e. the sum of the instantaneous squared errors). The gradient of the cost function with respect to a synaptic filter is expanded using the chain rule:

$$\frac{\partial C}{\partial \mathbf{w}_{ij}^l} = \sum_k \frac{\partial C}{\partial s_j^{l+1}(k)} \cdot \frac{\partial s_j^{l+1}(k)}{\partial \mathbf{w}_{ij}^l}, \tag{22}$$

where

$$s_j^{l+1}(k) = \sum_i s_{i,j}^{l+1}(k) = \sum_i \mathbf{w}_{i,j}^l \cdot \mathbf{x}_i^l(k) \tag{23}$$

specifies the input to neuron j in layer l at time k. Note this expansion differs from the traditional approach of writing the total gradient as the sum of instantaneous gradients: $\partial C/\partial s_j^{l+1}(k) \cdot \partial s_j^{l+1}(k)/\partial \mathbf{w}_{ij}^l \neq \partial e^2(k)/\partial \mathbf{w}_{ij}^l$. Only the sums over all k are equivalent.

From Eq. 22 a stochastic algorithm is formed:

$$\mathbf{w}_{ij}^{l}(k+1) = \mathbf{w}_{ij}^{l}(k) - \eta \frac{\partial C}{\partial s_j^{l+1}(k)} \cdot \frac{\partial s_j^{l+1}(k)}{\partial \mathbf{W}_{ij}^{l}}. \tag{24}$$

From Eq. 23 it follows immediately that $\partial s_j^{l+1}(k)/\partial \mathbf{w}_{ij}^{l} = \mathbf{x}_i^l(k)$ for all layers in the network. Defining $\partial C/\partial s_j^l(k) \equiv \delta_j^l(k)$ allows us to rewrite Eq. 24 in the more familiar notational form

$$\mathbf{w}_{ij}^{l}(k+1) \;\; = \;\; \mathbf{w}_{ij}^{l}(k) - \eta \delta_j^{l+1}(k) \cdot \mathbf{x}_i^l(k). \tag{25}$$

To show this holds for all layers in the network, an explicit formula for $\delta_j^l(k)$ must be found. Starting with the output layer, we have simply

$$\delta_j^L(k) \equiv \frac{\partial C}{\partial s_j^L(k)} = \frac{\partial e^2(k)}{\partial s_j^L(k)} = -2e_j(k) f'(s_j^L(k)). \tag{26}$$

where $e_j(k)$ is the error at an output node. For a hidden layer, we again use the chain rule, expanding over all time and all N_{l+1} inputs $s^{l+1}(k)$ in the next layer:

$$\begin{aligned}
\delta_j^l(k) &\equiv \frac{\partial C}{\partial s_j^l(k)} \\
&= \sum_{m=1}^{N_{l+1}} \sum_t \frac{\partial C}{\partial s_m^{l+1}(t)} \frac{\partial s_m^{l+1}(t)}{\partial s_j^l(k)} \\
&= \sum_{m=1}^{N_{l+1}} \sum_t \delta_m^{l+1}(t) \frac{\partial s_m^{l+1}(t)}{\partial s_j^l(k)} \\
&= f'(s_j^l(k)) \sum_{m=1}^{N_{l+1}} \sum_t \delta_m^{l+1}(t) \frac{\partial s_{jm}^{l+1}(t)}{\partial x_j^l(k)}.
\end{aligned} \tag{27}$$

But recall

$$s_{jm}^{l+1}(t) = \sum_{k'=0}^{T^l} w_{jm}^l(k') x_j^l(t-k'). \tag{28}$$

Thus

$$\frac{\partial s_{jm}^{l+1}(t)}{\partial x_j^l(k)} = \begin{cases} w_{jm}^l(t-k) & \text{for } 0 \le t-k \le T^l \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

which now yields

$$\begin{aligned}
\delta_j^l(k) &= f'(_j^l(k)) \sum_{m=1}^{N_{l+1}} \sum_{t=k}^{T^l+k} \delta_m^{l+1}(t) w_{jm}^l(t-k) \\
&= f'(s_j^l(k)) \sum_{m=1}^{N_{l+1}} \sum_{n=0}^{T^l} \delta_m^{l+1}(k+n) w_{jm}^l(n) \\
&= f'(s_j^l(k)) \cdot \sum_{m=1}^{N_{l+1}} \boldsymbol{\delta}_m^{l+1}(k) \cdot \mathbf{w}_{jm}^l,
\end{aligned} \tag{30}$$

where we have defined

$$\boldsymbol{\delta}_m^l(k) = \left[\delta_m^l(k), \delta_m^l(k+1), ..\delta_m^l(k+T^{l-1})\right]. \tag{31}$$

Summarizing, the complete adaptation algorithm can be expressed as follows:

$$\mathbf{w}_{ij}^l(k+1) = \mathbf{w}_{ij}^l(k) - \eta \delta_j^{l+1}(k) \cdot \mathbf{x}_i^l(k) \tag{32}$$

$$\delta_j^l(k) = \begin{cases} -2e_j(k)f'(s_j^L(k)) & l = L \\ f'(s_j^l(k)) \cdot \sum_{m=1}^{N_{l+1}} \boldsymbol{\delta}_m^{l+1}(k) \cdot \mathbf{w}_{jm}^l & 1 \le l \le L-1. \end{cases} \tag{33}$$

A Causality Condition: Careful inspection of the above equations reveals that the calculations for the $\delta_j^l(k)$'s are, in fact, noncausal. The source of this noncausal filtering can be seen by considering the definition of $\delta_j^l(k) = \partial C / \partial s_j^l(k)$. Since it takes time for the output of any internal neuron to completely propagate through the network, the change in the *total* error due to a change in an internal state is a function of future values within the network. Since the network is FIR, only a finite number of future values must be considered, and a simple reindexing allows us to rewrite the algorithm is a causal form:

$$\mathbf{w}_{ij}^{L-1-n}(k+1) = \mathbf{w}_{ij}^{L-1-n}(k) - \eta \delta_j^{L-n}(k-nT) \cdot \mathbf{x}_i^{L-1-n}(k-nT) \tag{34}$$

$$\delta_j^{L-n}(k-nT) = \begin{cases} -2e_j(k)f'(s_j^L(k)) & n = 0 \\ f'(s_j^{L-n}(k-nT)) \cdot \sum_{m=1}^{N_{l+1}} \boldsymbol{\delta}_m^{L+1-n}(k-nT) \cdot \mathbf{w}_{jm}^{L-n} & 1 \le n \le L-1. \end{cases} \tag{35}$$

While less aesthetically pleasing than the earlier equations, they differ only in terms of a change of indices. These equations are implemented by propagating the delta terms backward continuously *without* delay. However, by definition this forces the internal values of deltas to be shifted in time. Thus one must buffer the states $\mathbf{x}(k)$ appropriately to form the proper terms for adaptation. Added storage delays are necessary only for the states $\mathbf{x}(k)$. The backward propagation of the delta terms require no additional delay and is still symmetric to the forward propagation. The net effect of this is to delay the actual gradient update by a few time steps. This may result in a slightly different convergence rate and misadjustment as in the analogous linear *Delayed LMS* algorithm [41, 42].

For simplicity we have assumed that the order of each synaptic filter, T, was the same in each layer. This is clearly not necessary. For the general case, let T_{ij}^l be the order of the synaptic filter connecting neuron i in layer l to neuron j in the next layer. Then in Eq. 34 and 35 we simply replace nT by $\sum_{l=L-n}^{L-1} \max_{ij}\{T_{ij}^l\}$. The basic rule is that the time shift for the delta associated with a given neuron must be made equal to the total number of tap delays along the longest path to the output of the network.

6 Acknowledgments

I would like to thank Andreas Weigend and Neil Gershenfeld for organizing the Santa Fe Workshop, with special thanks to Andreas for his enthusiastic feedback and ever stimulating conversations. This work was sponsored in part by NASA under contract NCA2-680 and Navy contract N00014-86-K-0718.

References

[1] G. Yule, "On a method of investigating periodicity in disturbed series with special reference to Wolfer's sunspot numbers", Philos. Trans. Roy. Soc. London, A 226, pp. 267-298, 1927.

[2] F. Takens, "Detecting Strange Attractors in Fluid Turbulence", in *Dynamical Systems and Turbulence*, eds. D. Rand and L.-S. Young, Springer, Berlin, 1981

[3] N. Packard, J. Crutchfield, J. Farmer and R. Shaw, "Geometry from a Time Series", *Phys. Rev. Lett.*, 45, pp. 712-716, 1980.

[4] K. Hornik, M. Stinchombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol 2, pp. 359-366,1989.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, Vol. 2 No. 4, 1989.

[6] B. Irie and S. Miyake, "Capabilities of three-layered perceptrons", *Proceedings of the IEEE Second International Conference on Neural Networks*, Vol. I, San Diego, CA, July 1988, pp. 641-647.

[7] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", Ph.D. thesis, Harvard University, Cambridge, MA, 1974.

[8] P. Werbos, "Generalization of Backpropagation with Application to a Recurrent Gas Market Model", in *Neural Networks*, vol. 1, pp. 339-356, 1988.

[9] A Lapedes and R. Farber, "Nonlinear Signal Processing Using Neural Networks: Prediction and system modeling", Technical Report LA-UR-87-2662, Los Alamos National Laboratory, 1987.

[10] A. Weigend, B. Huberman, and D. Rumelhart, "Predicting the future: a connectionist approach", in *International Journal of Neural Systems*, vol. 7, no.3-4, pp. 403-430, 1990.

[11] E. Wan, "Temporal backpropagation for FIR neural networks," *International Joint Conference on Neural Networks*, Vol. 1, San Diego, 1990, pp.575-580.

[12] E. Wan, "Temporal backpropagation: an efficient algorithm for finite impulse response neural networks", Proceedings of the 1990 Connectionist Models Summer School, Morgan Kaufmann, pages 131-140, 1990.

[13] K. Lang, G. Hinton, "A time-delay neural network architecture for speech recognition", Tech Report CMU-CS-88-152, Carnegie-Mellon University.

[14] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 3, pp. 328-339, March 1989.

[15] D. E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Cambridge, MA: MIT Press, 1986.

[16] L. Ljung, *System Identification: Theory for the User*, Englewood Cliffs, NJ, Prentice-Hall, 1987.

[17] S. Wei and W. William, *Time Series Analysis: Univariate and Multivariate Methods*, Addison-Wesley, 1990.

[18] D. Junge, *Nerve and Muscle Excitation*, Third Edition, Sinauer Associates, Inc, 1991.

[19] C. Koch amd I. Segev editors, *Methods in Neuronal Modeling: From Synapses to Networks*, MIT Press, 1989.

[20] R. MacGregor and E. Lewis, *Neural Modeling*, Plenum Press, New York, 1977.

[21] B. Widrow and S. D. Stearns. (1985) *Adaptive Signal Processing,* Englewood Cliffs, NJ, Prentice Hall.

[22] A. Waibel, "Modular Construction of Time-Delay Neural Networks for Speech Recognition", *Neural Computation*, vol.1, no.1,pp. 39-46, Spring 1989.

[23] Y. LeCun, "Generalization and Network Design Strategies", in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, eds. North-Holland, Amsterdam, 1989.

[24] Y. LeCun, B. Boser, et al., "Backpropagation Applied to Handwritten Zip Code Recognition", in *Neural Computation*, Vol 1, pp. 541-551, winter 1989.

[25] J. Mendel, *Discrete Techniques of Parameter Estimation: The Equation Error Formulation*, Marcel Dekker, New York, 1973

[26] R. Gooch, "Adaptive pole-zero filtering: the equation error approach", Ph.D. diss. Stanford University, 1983.

[27] R. Williams, D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", in *Neural Computations*, vol.1, pp. 270-280, 1989.

[28] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma" in *Neural Computation*, Vol 4, No. 1, pp. 1-58, 1992.

[29] J. Shynk , "Adaptive IIR Filtering", in *IEEE ASSP Magazine*, pp. 4-21, April 1989.

[30] I. Landau, *Adaptive Control: The Model Reference Approach*, Marcel Dekker, New York, 1979.

[31] C. Johnson, Jr., "Adaptive IIR filtering: Current results and open issues", *IEEE Trans. Inform. Theory,* vol. IT-30, no.2, pp. 237-250, Mar. 1984.

[32] S. Stearns, "Error surfaces or recursive adaptive filters," *IEEE Trans. Circuits Systems*, vol. CAS-28, no. 6, pp.603-606, June 1981.

[33] U. Huebner, N.B. Abrahm, and C.O. Weiss, " Dimension and entropies of a chaotic intensity pulsations in a single-mode far-infrared NH3 laser," *Physics Review*, A 40, pp. 6354, 1989.

[34] *Proceedings of the NATO Advanced Research Workshop on Time Series Prediction and Analysis*, (Santa Fe, New Mexico, May 14-17 1992), edited by A. Weigend and N. Gershenfeld, Addison-Wesley, 1993.

[35] D. MacKay, "Bayesian Interpolation", in *Neural Computations*, vol 4, no. 3, pp. 415-447, May 1992.

[36] D. MacKay, "A Practical Bayesian Framework for Backpropagation Networks", in *Neural Computations*, vol 4, no. 3, pp. 448-472, May 1992.

[37] J. Skilling, D. Robinson, S. Gull, "Probabilistic displays", in *Maximum Entropy and Bayesian Methods, Laramie, 1990*, W Grandy and L. Schick, eds., pp.365-368. Kluwer, Dordrecht.

[38] J. Moody, "The *Effective* Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Systems", this volume.in *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[39] T. Kailath, "Linear Systems", Prentice-Hall.,Englewood Cliffs, N.J., 1980.

[40] A. Bryson, Y. Ho, "Applied Optimal Control", Hemisphere Publishing Corp. NY, 1975.

[41] P. Kabal, "The stability of adaptive minimum mean square error equalizers using delayed adjustment", in *IEEE Transactions on Communications*, Vol, Com-31, No. 3, pp 430-432, March 199.

[42] G. Long, F. Ling, J. Proakis, "The LMS algorithm with Delayed Coefficient Adaptation", in *IEEE Transactions on Acoustics, Speech, and Signal Processing.*, vol. 27, no. 9, pp. 1397-1405, Sept. 1989.