Neural Networks for Time Series Processing

Georg Dorffner
Dept. of Medical Cybernetics and Artificial Intelligence
University of Vienna
and *Austrian Research Institute for Artificial Intelligence*

Abstract

This paper provides an overview over the most common neural network types for time series processing, i.e. pattern recognition and forecasting in spatio-temporal patterns. Emphasis is put on the relationships between neural network models and more classical approaches to time series processing, in particular, forecasting. The paper begins with an introduction of the basics of time series processing, and discusses feedforward as well as recurrent neural networks, with respect to their ability to model non-linear dependencies in spatio-temporal patterns.

1 Introduction

The world is always changing. Whatever we observe or measure – be it a physical value such as temperature or the price of a freely traded good – is bound to be different at different points in time. Classical pattern recognition, and with it a large part of neural network applications, has mainly been concerned with detecting systematic patterns in an array of measurements which do not change in time (static patterns). Typical applications involve the classification of input vectors into one of several classes (discriminant analysis), or the approximate description of dependencies between observables (regression). When changes over time are also taken into account, an additional, temporal dimension is added. Although to a large extent such a problem can still be viewed in classical pattern recognition terms, several additional important aspects come into play. The field of statistics concerned with analysing such spatio-temporal data (i.e. data that has a spatial and temporal dimension) is usually termed *time series processing*.

This paper aims at introducing the fundamentals of using neural networks for time series processing. As a tutorial article it naturally can only

scratch the surface of this field and leave many important details untouched. Nevertheless, it provides an overview of the most relevant aspects which form the basis of work in this field. Throughout the paper, references are given as a guide to further, more detailed literature. Basic knowledge about neural networks, architectures, and learning algorithms is assumed.

2 Time series processing

2.1 Basics

In formal terms, a time series is a sequence of vectors, depending on time t:

$$\vec{x}(t), t = 0, 1, ... \tag{1}$$

The components of the vectors can be any observable variable, such as, for instance

- the temperature of air in a building
- the price of a certain commodity at a given stock exchange
- the number of births in a given city
- the amount of water consumed in a given community

Theoretically, $\vec{x}$ can be seen as a continuous function of the time variable t. For practical purposes, however, time is usually viewed in terms of discrete time steps, leading to an instance of $\vec{x}$ at every end point of a – usually fixed-size – time interval. This is why one speaks of a time *sequence* or *series*. The size of the time interval usually depends on the problem at hand, and can be anything from millseconds, hours to days, or even years.

In many cases, observables are available only at discrete time steps (e.g. the price of a commodity at each hour, or day) naturally giving rise to a time series. In other cases (e.g. the number of births in a city), values have to be accumulated or averaged over a time interval (e.g. to lead to the number of births per month) to obtain the series. In domains where time is indeed continuous (e.g. when temperature in a given place is the observable) one must measure the variable at points given through the chosen time interval (e.g. measuring the temperature at every full hour) to obtain a series. This is called *sampling*. The sampling frequency, i.e. the number of points measured resulting from the chosen time interval, is a crucial parameter in this case, since different frequencies can essentially change the main characteristics of the resulting time series.

It should be noted that there is another field very closely related to time series processing, namely *signal processing*. Examples are speech recognition, detection of abnormal patterns in electrocardiograms (ECGs), or the automatic staging of sleep-electroencephalograms (EEGs). A signal, when sampled into a sequence of values at discrete time steps, constitutes a time series as defined above. Thus there is no formal distinction between signal and time series processing. Differences can be found in the type of prevalent applications (e.g. recognition or filtering in signal processing; forecasting in time series processing), the nature of the time series (the time interval in a sampled signal is usually a fraction of a second, while in time series processing the interval often is from hours upwards.), etc. But this is only an observation in terms of prototypical applications, and no clear boundary can be drawn. Thus, time series processing can profit from exploring methods from signal processing, and vice versa. An overview of neural network applications in signal processing can be found, among others, in [54, 51].[1]

If the vector $\vec{x}$ contains only one component, which is the case in many applications, one speaks of a *univariate* time series, otherwise it is a *multivariate* one. It depends very much on the problem at hand whether a univariate treatment can lead to results with respect to recognizing patterns or systematicities. If several observables influence each other – such as the air temperature and the consumption of water – a multivariate treatment – i.e. an analysis based on several observables (more than one component in $\vec{x}$) – would be indicated. In most of the discussions that follow I will nevertheless concentrate on univariate time series processing.

2.2 Types of processing

Depending on the goal of time series analysis, the following typical applications can be distinguished:

1. forecasting of future developments of the time series
2. classification of time series, or a part thereof, into one of several classes
3. description of a time series in terms of the parameters of a model
4. mapping of one time series onto another

Application type 1 is certainly the most wide-spread and imminent in literature. From econometrics to energy planning a large number of time series problem involve the prediction of future values of the vector $\vec{x}$ – e.g. in order to decide upon a trading strategy or in order to optimize

[1] The latter reference is given as one example of an IEEE proceedings series, resulting from an annual conference on neural networks for signal processing.

production. Formally, the problem is described as follows: Find a function $\mathbf{F} : \mathcal{R}^{k \times n+l} \rightarrow \mathcal{R}^k$ (with k being the dimension of $\vec{x}$) such as to obtain an estimate $\hat{\vec{x}}(t+d)$ of the vector $\vec{x}$ at time $t+d$, given the values of $\vec{x}$ up to time t, plus a number of additional time-independent variables (exogenous features) π_i:

$$\hat{\vec{x}}(t+d) = \mathbf{F}(\vec{x}(t), \vec{x}(t-1), \ldots, \pi_1, \ldots, \pi_l) \tag{2}$$

d is called the *lag* for prediction. Typically, $d = 1$, meaning that the subsequent vector should be estimated, but can take any value larger than 1, as well (e.g. the prediction of energy consumption 5 days ahead). For the sake of simplicity, I will neglect the additional variables π_i throughout this paper. We should keep in mind, though, that the inclusion of such features (e.g. the size of the room a temperature is measured in) can be decisive in some applications.

Viewed this way, forecasting becomes a problem of *function approximation*, where the chosen method is to approximate the continuous-valued function $\mathbf{F}$ as closely as possible. In this sense, it can be compared to function approximation or regression problems involving static data vectors, and many methods from that domain can be applied here, as well (see, for instance, [18], for an introduction). This observation will turn out to be important when discussing the use of neural networks for forecasting.

Usually the evaluation of forecasting performance is done by computing an error measure E over a number of time series elements, such as a validation or test set:

$$E = \sum_{i=0}^{N} e(\hat{\vec{x}}(t-i), \vec{x}(t-i)) \tag{3}$$

e is a function measuring a single error between the estimated (forecast) and actual sequence element. Typically, a distance measure (Euclidean or other) is used here, but depending on the problem, any funtion can be used (e.g. a function computing the cost resulting from forecasting $\vec{x}(t+d)$ incorrectly).

In many forecasting problems, the exact value of $\hat{\vec{x}}(t+d)$ is not required, only an indication of whether $\vec{x}(t+d)$ will be larger (rising) or smaller (falling) than $\vec{x}(t)$, or remain approximately the same. If this is the case, the problem turns into a classification problem, mapping the sequence (or a part thereof) onto the classes *rising* or *falling* (and perhaps *constant*).

In more general terms, classification of time series (application type 2 above) can be expressed as the problem of finding a function $\mathbf{F}_c : \mathcal{R}^{k \times n+l} \rightarrow \mathcal{B}^k$ assigning one out of several classes to a time series:

$$\mathbf{F}_c : (\vec{x}(t), \vec{x}(t-1), \ldots, \pi_1, \ldots, \pi_l) \rightarrow \hat{c}_i \in \mathcal{C} \tag{4}$$

where $\mathcal{C}$ is the set of available class labels. Formally, there is no essential difference to the function approximation problem (equation 2). In other words, classification can be viewed as a special case of function approximation, where the function to be approximated maps continuous vectors onto binary-valued ones. A difference comes from the way in which the problem is viewed (i.e. a separation of vectors is sought rather than an approximation of the dependencies between them) – which can have an influence on what method to derive the function is used – and from the way performance is evaluated. Typically, an error function takes on the form

$$E = 1 - \frac{1}{N} \sum_{i=1}^{N} \delta_{\hat{c}_i, c_i} \tag{5}$$

expressing the percentages of inputs whch are not correctly assigned the desired class. δ_{ij} is the Kronecker symbol, i.e. $\delta_{ij} = 1$ iff $i = j$, and 0 otherwise. c_i is the known class label of input i. Again, this distinction between approximation (regression) and classification (discrimination) is the same as in pattern recognition of vectors without temporal dimension. Thus, a large number of results and methods from that domain can be used for time series classification, as well. Another difference in the domain of time series processing is that classification (with the exception of a classification into *rising/falling*) usually is *retrospective* – i.e. there is no time lag for the estimated output – rather than *prospective* (forecast into the future).

Application type 3 – modeling of time series – is implicitly contained in most instances of 1 (forecasting) and 2 (classification). The function $\mathbf{F}$ in equation 2 can be considered as a *model* of the time series which is capable of *generating* the series, by successively substituting inputs by estimates. To be useful, a model should have fewer parameters (degrees-of-freedom in the estimation of $\mathbf{F}$) than elements in the time series. Since the latter number is potentially infinite, this basically means that the function $\mathbf{F}$ should depend only on a finite and fixed number of parameters (which, as we will see below, does *not* mean that it can only depend on a bounded number of past sequence elements). Besides its use in forecasting and classification, a model can also be used as a description of the time series, its parameters being viewed as a kind of *features* of the series, which can be used in a subsequent analysis (e.g. a subsequent classification, together with time-independent features). This can be compared with the process of modeling with the aim of *compressing* data vectors in the purely spatial domain (e.g. by realising an auto-associative mapping with a neural network, [13]).

Finally, while modeling is a form of mapping a time series onto itself (i.e. finding model parameters based on the time series in order to reproduce the series), the mapping of one time series onto another, different, one is conceivable as well (application type 4). A simple example would be

forecasting the value of one series (e.g. the price of oil) given the values of another (e.g. interest rates). More complex applications could involve the separate modeling of two time series and finding a functional mapping between them. Since in its simplest form, mapping between time series is a special case of mulivariate time series processing (discussed above), and in the more complex case it is not very common, this application type will not be discussed further. State-space models, however (to be discussed below), can be viewed in this context.

In what follows, I will mainly discuss forecasting problems, while keeping in mind that the other application types are very closely related to this type.

2.3 Stochasticity of time series

The above considerations implicitly assume that theoretically an exact model of a time series can be found (i.e. one that minimizes the error measure to any desired degree). For real-world applications, this assumption is not realistic. Due to measuring errors and unknown or uncontrollable influencing factors, one almost always has to assume that even the most optimal model will lead to a *residual* error ϵ which cannot be erased. Usually, this error is assumed to be the result of a noise process, i.e. produced randomly by an unknown source. Therefore, equation 2 has to be extended as following:

$$\vec{x}(t+d) = \mathbf{F}(\vec{x}(t), \vec{x}(t-1), ...) + \vec{\epsilon}(t) \tag{6}$$

This noise $\vec{\epsilon}(t)$ cannot be included into the model explictly. However, many methods assume a certain characteristic of the noise (e.g. Gaussian white noise), the main describing parameters of which (e.g. mean and standard deviation) *can* be included in the modeling process. By doing this in forecasting, for instance, one cannot only give an estimate of the forecast value, but also an estimate of how much this value will be disturbed by noise. This is the focus of so-called ARCH models [4].

2.4 Preprocessing of time series

In only a few cases it will be appropriate to use the measured observables immediatly for processing. In most cases, it is necessary to pre-analyze, as well as preprocess the time series to ensure an optimal outcome of the processing. One one hand, this has to do with the method employed, which can usually extract only certain kinds of systematicities (i.e. usually those that are expressed in terms of vector similarities). On the other hand, it is necessary to remove known systematicities which could hamper the performance. An example are clear (linear or non-linear) *trends*, i.e. the phenomenon that the average value of sequence elements is constantly rising

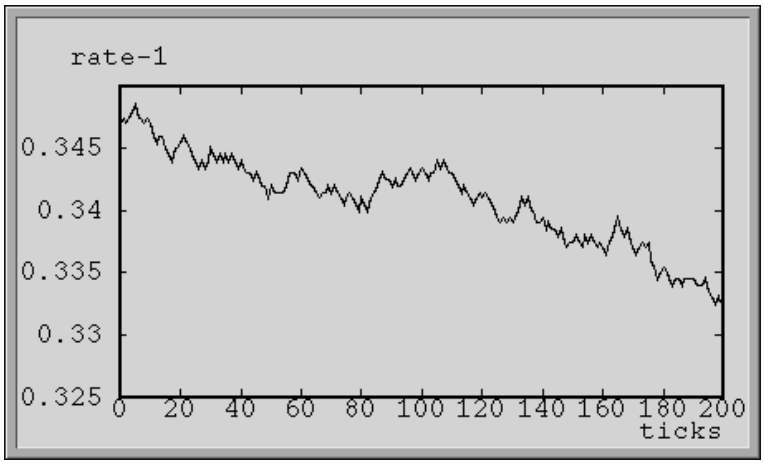

Figure 1: A time series showing a close-to-linear falling trend. The series consists of tick-by-tick currency exchange rates (Swiss franc - US$). Source: ftp://ftp.cs.colorado.edu/pub/Time-Series/SantaFe

or falling (see figure 1, taken from [50]). By replacing the time series $\vec{x}(t)$ with a series $\vec{x}'(t)$, consisting of the differences between subsequent values,

$$\vec{x}'(t) = \vec{x}(t) - \vec{x}(t-1) \tag{7}$$

a linear trend is removed (see figure 2). This differencing process corresponds to differentiation of continuous functions. Similarly, *seasonalities*, i.e. periodic patterns due to a periodic influencing factor (e.g. day of the week in product sales), can be eliminated by computing the differences between corresponding sequence elements:

$$\vec{x}'(t) = \vec{x}(t) - \vec{x}(t-s) \tag{8}$$

(e.g. $s = 7$ if the time interval are days, and corresponding days of the week show similar patterns).

Identifying trends and seasonalities, when they are a clearly visible property of the time series, lead to prior knowledge about the series. Like for any statistical problem, such prior knowledge should be handled explicitely (by differencing, and summation after processing to obtain the original values). Otherwise any forecasting method will mainly attempt to model these perspicuous characteristics, leaving little or no room for the more fine-grained characteristics. (Thus a naive forecaster, e.g. "forecast today's value plus a constant increment" will probably fare equally well.) For non-linear trends, usually a parametric model (e.g. an exponential curve) is assumed and etsimated, based on which an elimination can be done, as well.

Another reason for eliminating trends and seasonalities (or, for that matter, any other clearly visible or well-known pattern) is that many methods

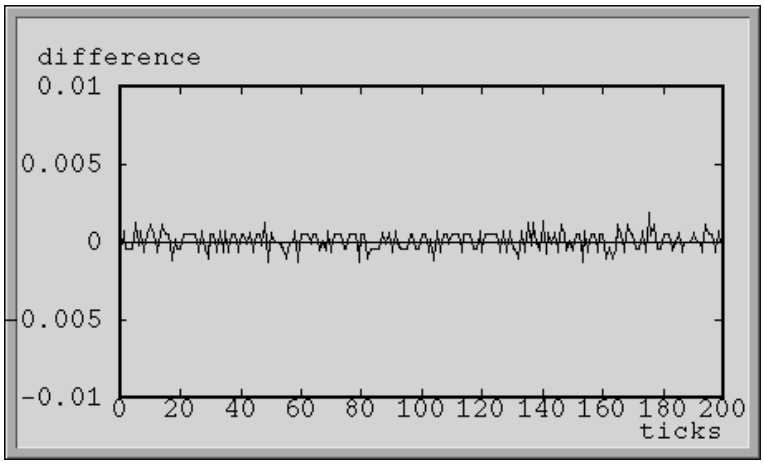

Figure 2: The time series from the previous figure, after differencing

require *stationarity* of the time series (more on this below).

3 Neural nets for time series processing

Several authors have given an overview of different types of neural networks for use in time series processing. [47], for instance, distinguishes different neural networks according to the type of mechanism to deal with temporal information. Since most neural networks have previously been defined for pattern recognition in static patterns, the temporal dimension has to be supplied in an appropriate way. [47] distinguishes the following mechanisms:

- layer delay without feedback (or time windows)
- layer delay with feedback
- unit delay without feedback
- unit delay with feedback (self-recurrent loops)

[33] bases his overview on a distinction concerning the type of memory: delay (akin to time windows and delays), exponential (akin to recurrent connections) and gamma (a memory model for continuous time domains).

I would like to give a slightly different overview. Given the above discussion of time series processing, the use of neural networks in this field can mainly be seen in the context of function approximation and classification. In the following, the main neural network types will be introduced and discussed along more traditional ways of sequence processing.

Other introductions can be found in [45], [41], [10], and [23]. Extensive treatments of neural networks for sequence processing are the book by [3].

4 Mulilayer perceptrons and radial basis function nets: autoregressive models

Among the most wide-spread neural networks are feedforward networks for classification and function approximation, such as multilayer perceptrons (MLP; hidden units with sigmoidal transfer functions, [42]) and radial basis function networks (RBFN; hidden units using a distance propagation rule and a Gaussian, or other, transfer functions, [8]). Both network types have been proven to be universal function approximators (see [14, 25] for the MLP, and [30, 22] for the RBFN). This means that they can approximate any reasonable function $\mathbf{F}(\vec{p}) : \mathcal{R}^n \rightarrow \mathcal{R}^m$ arbitrarily closely by

$$\mathbf{F}^{MLP}(\vec{p}) = \left(\sum_{j=1}^{k} v_{jl} \sigma(\sum_{i=1}^{n} w_{ij} p_i - \theta_j) - \theta_l \right), l = 1..m \tag{9}$$

– where σ is the sigmoid function (or any other non-linear, non-polynomial function), k is the number of hidden units, v_{jl} and w_{ij} are weights, and θ_i are thresholds (biases) – or by

$$\mathbf{F}^{RBF}(\vec{p}) = \left(\sum_{j=1}^{k} v_{jl} \Gamma(\sum_{i=1}^{n} (w_{ij} - p_i)^2) - \theta_l \right), l = 1..m \tag{10}$$

where Γ is the Gaussian function, provided k is sufficiently large. Approximation of non-linearity is done by a superposition of several instances of the basis function (e.g., sigmoid or Gaussian). With a fixed number of hidden units (as is the case in most neural network applications) the method could be called a *semi-parametric* approximation of functions: It does not make specific assumptions about the shape of the function (as would a *parametric* method), but it cannot approximate any arbitrarily complex function (as could a *non-parametric* techniqu; note that we assumed a fixed number of hidden units, while the above proofs require an arbitrarily large number of units, not fixed beforehand) – see, for instance, [18] or [5].

From this observation, MLPs and RBFNs offer a straight-forward extension to a wide-spread classical way of modeling time series: linear autoregressive models. Linear autoregresive time series modeling (see [7]) assumes the function $\mathbf{F}$ in equation 2 to be a linear combination of a fixed number of previous series vectors.[2] Including the noise term ϵ,

$$x(t) = \sum_{i=1}^{p} \alpha_i x(t-i) + \epsilon(t) \tag{11}$$

[2]for simplification, a univariate series is assumed, by replacing the vectors $\vec{x}$ with scalars x

$$= F^L(x(t-1), \ldots, x(t-p)) + \epsilon(t) \quad (12)$$

If p previous sequence elements are taken, one speaks of an AR[p] model of the time series (autoregressive model of order p). Finding an appropriate AR[p] model means choosing an appropriate p and estimating the coefficients α_i, e.g. through a least squares optimization procedure (see [7] for an extensive treatment of this topic). This technique, although rather powerful, is naturally limited, since it assumes a linear relationship among sequence elements. Most importantly, it also assumes *stationarity* of the time series, meaning that the main moments (mean, standard deviation) do not change over time (i.e. mean and standard deviation over a part of the series are independent of where in the series this part is extracted).

It becomes clear from equations 9 and 11 (or 10 and 11, respectively) that an MLP or RBFN can replace the linear function F^L in equation 11 by an arbitrary non-linear function F^{NN} (with NN being either MLP or RBF):

$$x(t) = F^{NN}(x(t), \ldots, x(t-p)) + \epsilon(t) \quad (13)$$

This non-linear function can be estimated based on samples from the series, using one of the well-known learning or optimization techniques for these networks (e.g. backpropagation, conjugent gradient, etc.). Making F^{NN} dependent on p previous sequence elements is identical to using p input units being fed with p adjacent sequence elements (see Fig. 3). This input is usually refered to as a *time window* (see section 3), since it provides a limited view on part of the series. It can also be viewed as a simple way of transforming the temporal dimension into another spatial dimension.

Non-linear autoregressive models are potentially more powerful than linear ones in that

- they can model much more complex underlying characteristics of the series
- they theoretically do not have to assume stationarity

However, as in static pattern recognition, they require much more care and caution than linear methods in that they

- require large numbers of sample data, due to their large number of degrees-of-freedom
- can run into a variety of problems, such as overfitting, sub-optimal minima as a result of estimation (learning), etc., which are more severe than in the linear case (where overfitting can come about by chossing too high a value for the parameter p, for instance)

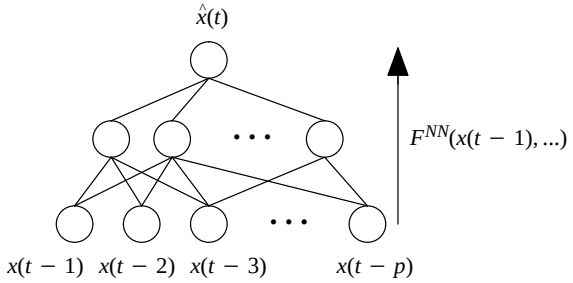

Figure 3: A feedforward neural net with time window as a non-linear AR model

- do not necessarily include the linear case in a trivial way

Especially the first point is important for many real-world applications where only limited data is available. A linear model might still be preferable in many cases, even if the dependencies are non-linear. The second point concerns the learning algorithm employed. Backpropagation very often is not the most appropriate choice to obtain optimal models.

Examples of feedforward neural networks in forecasting are [9, 36, 49, 53] and numerous other papers in [44] and [37].[3]

4.1 Time-delay neural networks

Another mechanism to supply neural networks with "memory" to deal with the temporal dimension is the introduction of time delays on connections. In other words, through delays, inputs arrive at hidden units at different points in time, thus being "stored" long enough to influence subsequent inputs. This approach, called a *time-delay neural network (TDNN)* has been extensively employed in speech recognition, for instance, by [52]. Formally, time delays are identical to time windows and can thus be viewed as autoregressive models, as well. An interesting extension is the introduction of time delays also on connections between hidden and output units, providing additional, more "abstract" memory to the network.

5 "Jordan" nets: moving average models

An alternative approach to modeling time series is to assume the series being generated through a linear combination of q "noise" signals (see, again, [7]):

[3]This, again, is one example of a proceedings series, resulting from the annual conference "Neural Networks and the Capital Markets", providing an excellent overview of work on forecasting with neural networks in the financial domain.

$$x(t) = -\sum_{i=1}^{q} \beta_i \epsilon(t-i) + \epsilon(t) \quad (14)$$

$$= F^L(\epsilon(t-1), \ldots, \epsilon(t-q)) + \epsilon(t) \quad (15)$$

This is refered to as a *moving average* (or MA[q]) model ("of order q"). The approach seems paradoxical at first: a non-random time series is modeled as the linear combination of random signals. However, when viewing the linear combination as a discrete filter of the noise signal, the MA[q] model can be viewed as thus: A noise process usually has a frequency spectrum containing all or a large number of frequencies ("white" noise). A filter – like the MA[q] model – can thus cut out any desired frequency spectrum (within the bounds of linearity), leading to a specific, non-random time series.

A combination of AR and MA components is given in the so-called ARMA[p,q] model:

$$x(t) = \sum_{i=1}^{p} \alpha_i x(t-i) - \sum_{i=1}^{q} \beta_i \epsilon(t-i) + \epsilon(t) \quad (16)$$

$$= F^L(x(t-1), \ldots, x(t-p), \epsilon(t-1), \ldots, \epsilon(t-q)) + \epsilon(t) \quad (17)$$

MA[q] and ARMA[p,q] models, like AR[p] models, are again rather limited given their linearity, and also their requirement of stationarity. Thus, an extension to the non-linear case using neural networks seems appropriate here, as well. [12] introduce such a possibility. The most important question to answer is this: What values of ϵ_i should be taken? A common approach in MA modeling is to use the difference between actual and estimated (forecast) value as an estimate of the noise term at time t. This is justified by the following observation. Assume that the model is already near-optimal in terms of forecasting. Then the difference between forecast and actual value will be close to the residual error – the noise term in equation 6. Thus, this difference can be used as an estimate $\hat{\epsilon}$ for the noise term ϵ in equation 16.

$$\hat{\epsilon}(t) = x(t) - \hat{x}(t) \quad (18)$$

Figure 4 depicts a neural network realizing this assumption for the univariate case [12]. The output of the network, which is identical to the estimate of $\hat{x}(t+1)$, is fed back to an additional input layer, each unit of which also receives a negative version of the corresponding actual value $x(t+1)$ (available at the subsequent time step) to form the desired difference. If a time window (or time delay at the input layer) is introduced, as well, the network forms an arbitrarily non-linear ARMA[p,q] model of the time series:

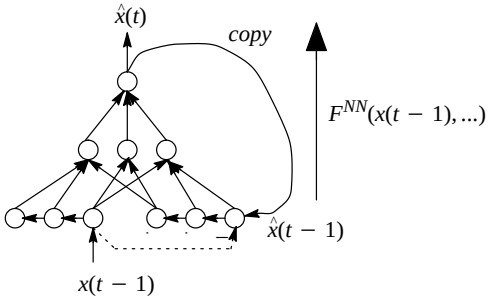

Figure 4: A neural network with output layer feedback, realizing a non-linear ARMA model.

$$x(t) = F^{NN}(x(t-1), \ldots, x(t-p), \hat{\epsilon}(t-1), \ldots, \hat{\epsilon}(t-q)) + \epsilon(t) \qquad (19)$$

Simliar observations as with respect to the non-linear AR[p] model in section 11 must be made. As a non-linear model, the network is potentially more powerful than traditional ARMA models. However, this must again be considered with care, due to the large numbers of degrees-of-freedom and the potential limitations of the learning algorithms. A further complication comes from the fact that at the beginning of the sequence, no estimates $\hat{x}$ are available. One possible way to overcome this problem is to start with 0 values and update the network until sufficient estimates are computed. This requires a certain number of cycles before the learning algorithm can be applied, "wasting" a number of sequence elements which cannot be used for training. This is especially important when one wants to randomize learning by always choosing an arbitrary window from the time series, instead of stepping thorugh the series sequentially.

The network in figure 4 can be considered a special case of the recurrent network type in figure 5, usually called *Jordan network* after [26]. It consists of a multilayer perceptron with one hidden layer and a feedback loop from the output layer to an additional input (or *context*) layer. In addition, [26] introduced self-recurrent loops on each unit in the context layer, i.e. each unit in the context layer is connected with itself, with a weight v_i smaller than 1. Without such self-recurrent loops, the network forms a non-linear function of p past sequence elements and q past estimates:

$$\hat{x}(t) = F^{NN}(x(t-1), \ldots, x(t-p), \hat{x}(t-1), \ldots, \hat{x}(t-q)) \qquad (20)$$

The non-linear ARMA[p,q] model discussed above can be said to be implicitly contained in this network by reformulating equation 20 (with the help of equation 9 and $\vec{p} = (x(t-1), \ldots, x(t-p), \hat{x}(t-1), \ldots, \hat{x}(t-q))$ as

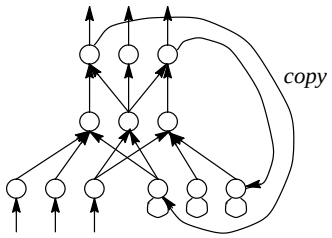

Figure 5: The "Jordan" network.

$$\hat{x}(t) =$$

$$\sum_{j=1}^{k} v_{jl}\sigma(\sum_{i=1}^{p}(w_{ij}\underbrace{(x(t-i)-\hat{x}(t-i))}_{\hat{\epsilon}(t-i)}+(w_{i+p,j}+w_{ij})\hat{x}(t-i)-\theta_j)-\theta_l$$

$$= \quad \overline{F^{NN}}(x(t-1),\ldots,x(t-p),\hat{\epsilon}(t-1),\ldots,\hat{\epsilon}(t-q)) \tag{21}$$

$$l=1 \tag{22}$$

provided, $p = q$ (a similar derivation can be made for $q < p$). However, conventional learning algorithms for MLPs cannot trivially recogize differences between input values (in terms of them being the relevant invariances). Therefore, the explicit calculation of the differences in figure 4 can be viewed as the inclusion of essential pre-knowledge and thus the above network seems to be more well-suited for the implementation of a clean non-linear ARMA model. Nevertheless, the Jordan network can also be used for time series processing, extending the ARMA family of models by one realizing a functional dependency between sequence elements and estimates one one hand, and the to-be-forecast value on the other. Examples of applications with "Jordan" networks are [16, 31].

The self-recurrent loops in the Jordan network are another deviation of the standard ARMA-type of models. With their help, past estimates are superimposed onto each other in the following way:

$$a_i^C(t) = f(a_i^C(t-1) + v_i\hat{x}_i(t-1)) \tag{23}$$

where f is the activation function, typically a sigmoid. This means that the activations a_i^C of the units in the context layer are recursively computed based on all past estimates $\hat{x}$. In other words, each such activation is a function of *all* past estimates and thus contains information about a potentially unlimited previous history. This property has often given rise to the argument that recurrent networks can exploit information beyond a limited time window (p or q past values). However, in practice this cannot really be exploited. If v_i is close to 1, the unit (if it uses a sigmoid transfer

function) quickly saturates to maximum activation, where additional inputs have little effect. If $v_i << 1$, the influence of past estimates quickly goes to 0 through several applications of equation 23. So, in fact, context layers with self-recurrent loops are also rather limited in representing past information. In addition, flexibility in including past information is paid by the loss of explicitness of that information, since past estimates are accumulated into one activation value.

Another way of employing self-recurrent loops will be discussed below.

6 Elman networks and state space models

Another common method for time series processing are so-called (linear) *state space models* [11]. The assumption is that a time series can be described as a linear transformation of a time-dependent state – given through a state vector $\vec{s}$:

$$\vec{x}(t) = \mathbf{C}\vec{s}(t) + \vec{\epsilon}(t) \tag{24}$$

where $\mathbf{C}$ is a transformation matrix. The time-dependent state vector is usually also desribed by a linear model:

$$\vec{s}(t) = \mathbf{A}\vec{s}(t-1) + \mathbf{B}\vec{\eta}(t) \tag{25}$$

where $\mathbf{A}$ and $\mathbf{B}$ are matrices, and $\vec{\eta}(t)$ is a noise process, just like $\vec{\epsilon}(t)$ above. The model for the state change, in this version, is basically an ARMA[1,1] process. The basic assumption underlying this model is the so-called *Markov assumption*, meaning that the next sequence element can be predicted by the state a system producing a time series is in, no matter how the state was reached. In other words, all the history of the series necessary for producing a sequence element can be expressed by one state vector. Since this vector ($\vec{s}$) is continuous-valued, all possible state vectors form a Euclidean *vector space* in $\mathcal{R}^n$. This model [11] can be viewed as a time series modeled in terms of another one (related to the mapping between time series discussed in section 2.2).

If we further assume that the states are also dependent on the past sequence vector (an assumption, which is common, for instance, in signal processing – see [24]), and neglect the moving average term $\mathbf{B}\vec{\eta}(t)$:

$$\vec{s}(t) = \mathbf{A}\vec{s}(t-1) + \mathbf{D}\vec{x}(t-1) \tag{26}$$

then we basically obtain an equation describing a recurrent neural network type, known as *Elman network* (after [19]), depicted in figure 6. The Elman network is an MLP with an additional input layer, called the *state layer*, receiving as feedback a copy of the activations from the hidden layer at the previous time step. If we use this network type for forecasting, and

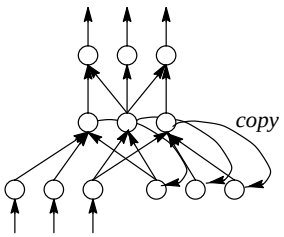

Figure 6: The "Elman" network as an instantiation of the state-space model.

equate the activation vector of the hidden layer with $\vec{s}$, the only difference to equation 26 is the fact that in an MLP a sigmoid activation function is applied to the input of each hidden unit:

$$\vec{s}(t) = \sigma(\mathbf{A}\vec{s}(t-1) + \mathbf{D}\vec{x}(t-1)) \tag{27}$$

where $\sigma(\vec{a})$ refers to the application of the sigmoid (or logistic) function $1/(1+\exp(-a_i))$ to each element a_i of $\vec{a}$. In other words, the transformation is not linear but the application of a *logistic regressor* to the input vectors. This leads to a restriction of the state vectors to vectors within a unit cube, with non-linear distortions towards the edges of the cube. Note, however, that this is a very restricted non-linear transformation function and does *not* represent the general form of non-linear state space models (see below).

The Elman network can be trained with any learning algorithm for MLPs, such as backpropagation or conjugent gradient. Like the Jordan network, it belongs to the class of so-called *simple recurrent networks (SRN)*[23]. Even though it contains feedback connections, it is not viewed as a dynamical system in which activations can spread indefinitly. Instead, activations for each layer are computed only once at each time step (each presentation of one sequence vector).

Like above, the strong relationship to classical time series processing can be exploited to introduce "new" learning algorithms. For instance, in [56] the *Kalman filter algorithm*, developed for the original state space model is applied to general recurrent neural networks.

Similar observations can be made about the Elman recurrent network as with respect to the Jordan net: Here, too, a number of time steps is needed until – after starting with 0 activations – suitable activations are available in the state layer, before learning can begin. Standard learning algorithms like backpropagation, although easy to apply, can cause problems or lead to non-optimal solutions. Finally, this type of recurrent net also cannot really deal with an arbitrarily long history, for similar reasons as above (see, for instance, [2], cited in [3], or [33]). Examples of applications with "Elman" networks are [21, 15, 17].

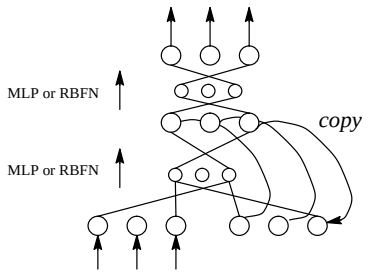

Figure 7: An extension of the "Elman" network as realization of a non-linear state-space model

As hinted upon above, a general non-linear version of the state space model is conceivable, as well. By replacing the linear transformation in equations 24 and 25 by an arbitrary non-linear function, one obtains

$$\vec{x}(t) = \mathbf{F}_1(\vec{s}(t)) + \vec{\epsilon}(t) \tag{28}$$

$$\vec{s}(t) = \mathbf{F}_2(\vec{s}(t-1)) + \vec{\eta}(t) \tag{29}$$

Like in the previous sections on non-linear ARMA models, these non-linear functions $\mathbf{F}_1$ and $\mathbf{F}_2$ could be modeled by an MLP or RBFN, as well. The resulting network is depicted in figure 7. An example of the application of such a network is [27].

6.1 Multi-recurrent networks

[46], and [47], has given an extensive overview of additional types of recurrencies, time-windows and time delays in neural networks. By combining several types of feedback and delay one obtains the general *multirecurrent network (MRN)*, depicted in figure 8. First, feedback from hidden *and* output layers are permitted. From the discussions in sections 4, 5 and 6 it becomes clear that his can be viewed as a state space model, where the state transition is modeled as a kind of ARMA[1,1] process, reintroducing the $\mathbf{B}\vec{\eta}(t)$ term in equation 25. This view is not entirely correct, though. Using the estimates $\hat{\vec{x}}$ as additional inputs implicitly introduces estimates for the noise process $\epsilon(t)$ in equation 24, and not for $\eta(t)$ in equation 25.

Secondly, all input layers (the actual input, the state and the context layer) are permitted to be extended by time-delays, such as to introduce time windows over past instances of the corresponding vectors. This essentially means that the involved processes are AR[p] and ARMA[p,q], respectively, with p and q larger than 1.

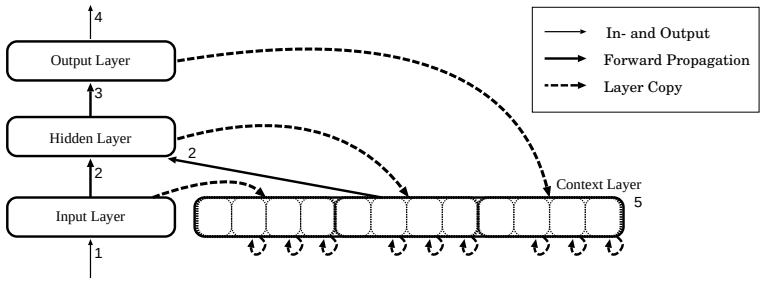

Figure 8: The multi-recurrent network from Ulbricht (1995).

Thirdly, like in the Jordan network, self-recurrent loops in the state layer can be introduced. The weights of these loops, and the weights of the feedback copies resulting from the recurrent one-to-one connections, are chosen such as to scale the theoretically maximum input to each unit in the state layer to 1, and to give more or less weight to the feedback connections or self-recurrent loops, respectively. If, for instance, 75 % of the total activation of a unit in the state layer comes from the hidden layer feedback, and 25 % comes from self-recurrency, the state vector will tend to change considerably at each time step. If, on the other hand, only 25 % come from the hidden layer feedback, and 75 % from the self-recurrent loops the state vector will tend to remain similar to the one at the previous time step. [47] speaks of *flexible* and *sluggish* state spaces, respectively. By introducing several state layers with different such weighting schemes, the network can exploit both the information of rather recent time steps and a kind of average of several past time steps, i.e. a longer, averaged history.

It is clear that a full-fletched version of the MRN contains a very large number of degrees-of-freedom (weights) and requires even more care than the other models discussed above. Several empirical studies [46, 48] have shown, however, that for real-world applications, some versions of the MRN can significantly outperform most other, more simple, forecasting methods. The actual choice of feedback, delays and weightings still depends largely on empirical evaluations, but similar iterative estimation algorithms as were suggested by [7] (for obtaining appropriate parameter values for ARMA models) appear applicable here, too.

Another advantage of self-recurrent loops becomes evident in applications where patterns in the time series can vary in time scale. This phenomenon is called *time warping*, and is especially known in speech recognition, where different speech patterns can vary in length and relationships between segments dependent on speaking speed and intonation [32]. In autoregressive models with fixed time windows, such distorted patterns lead to vectors that do not share sufficient similarities to be classified correctly. This is sometimes called the *temporal invariance problem* – the problem of

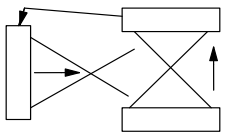

Figure 9: A simple network with a time-dependent weight matrix, produced by a second neural net.

recognizing temporal patterns independent of their duration and temporal distortion. In a state space model, implemeted as a recurrent network with self-recurrent loops such invariances can be dealt with, especially when sluggish state spaces are employed. If states in the state space model are forced to be similar at subsequent time steps, events can be treated equally (or similarly) even when the are shifted along the temporal dimension. This property is discussed extensively in [47].

7 Neural nets producing weight matrices: time-dependent state transitions

The original state space model approach (equations 24 and 25) left open the possibility of making all matrices **A** through **C** time-dependent as well. This allows for the modeling of non-stationary time series and series where the variance of the noise process changes over time. In neural network terms this would mean the introduction of time-varying weight matrices[4].

[34] introduced a small neural network model that can be viewed in the context of time-dependent transition matrices. It consist of two feedforward networks – one mapping an input sequence onto an output, and another one producing the weight matrix for the first network (figure 9). Even though not a state-space model but rather a AR[1] model of the input sequence, it realizes a mapping with a variable matrix (the weight matrix of the first network). This network was used to learn formal languages, such as parity or others. A similar example can be found in [43].

In this context, other approaches to inducing finite-state automata into neural networks should be mentioned (e.g. [20]). A finite state automaton is another classical model to describe time series, although on a more abstract level – the level of categories instead of continuous-valued input. If a categorization process is assumed before the model is applied it can also be used for time series as the ones discussed above. An automaton is defined as a set of states (discrete and finite, so there is no concept of state *space*

[4]By 'time-varying' I mean varying on the time-scale of the series. Weight changes due to learning are not considered here.

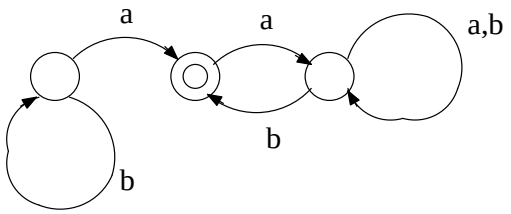

Figure 10: A finite state automaton for modeling time series

in this model) with arcs between them corresponding to state transitions taken dependent on the input. For instance, in figure 10, if – starting from the left-most state (node) – an 'a' is encountered, the automaton would jump to state 2, while if a 'b' is encountered, it would remain in state 1. Depending on what types of arcs emanate from a state, a prediction can be made with respect to what input element must follow, provided that the input is "grammatical", i.e. corresponds to the grammar the automaton implements. This is especially useful for sequences in speech or language.

[20] have shown that a neural network can be set or trained such as to implement such an automaton. While this concept cannot directly be applied to real-world sequences like the ones above, it points to another useful application of neural networks, especially when going from finite, discrete states to continuous state spaces. This is exactly what an Elman network or the model by [34] realizes.

8 Other topics

The story does not end here. There are many more important topics concerning time series processing and the use of neural networks in this field. Some topics that could not be covered here, but are of equal importance as the ones that were, are the following.

- many time series applications are tackled with *fully recurrent networks*, or networks with recurrent architectures different from the ones discussed (e.g. [39]). Special learning algorithms for arbitrary recurrent networks have been devised, such as *backpropagation in time*[42] and *real-time recurrent learning (RTRL)*[55].

- many authors use a combination of neural networks with so-called *hidden Markov models (HMM)* for time series and signal processing. HMMs are related to finite automata and describe probabilities for changing from one state to the other. See, for instance, [6] or the treatment in [3].

- unsupervised neural network learning algorithms, such as the self-organizing feature map, can also be applied in time series processing, both in forecasting [1] and classification [40]. The latter application constitutes an instance of so-called *spatio-temporal clustering*, i.e. the unsupervised classification of time series into clusters – in this case the clustering of sleep-EEG into sleep stages.

- a number of authors have investigated the properties of neural networks viewed as dynamical systems, including chaotic attractor dynamics. Examples are [28] and [35].

The focus of this paper was to introduce the most widely used architectures and their close relationships to more classical approaches to time series processing. The approaches presented herein can be viewed as starting points for future research, since the potential of neural networks – especially with respect to dynamical systems – is by far not fully exploited yet.

9 Conclusion

As mentioned initially, this overview of neural networks for time series processing could only scratch the surface of a very lively and important field. The paper has attempted to introduce most of the basics of this domain, and to stress the relationship between neural networks and more traditional statistical methodologies. It underlined one important contribution of neural networks – namely their elegant ability to approximate arbitrary non-linear functions. This property is of high value in time series processing and promises more powerful applications, especially in the subfield of forecasting, in the near future. However, it was also emphasized that non-linear models are not without problems, both with respect to their requirement for large data bases and careful evaluation and with respect to limitations of learning or estimation algorithms. Here, the relationship between neural networks and traditional statistics will be essential, if the former is to live up to the promises that are visible today.

Acknowledgments

The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Science, Research, and the Arts. I particularly thank Claudia Ulbricht, Adrian Trapletti and the research group of Prof. Manfred Fischer, the cooperation with which has been the basis for this work, for valuable comments on this manuscript.

References

[1] Baumann T., Germond A.J.: Application of the Kohonen Network to Short-Term Load Forecasting, in Proc. PSCC, Avignon, Aug 30 - Sep 3, 1993.

[2] Bengio Y., Simard P., Frasconi P.: Learning long-term dependencies with gradient descent is difficult, *IEEE Trans. Neural Networks* **5(2)**, 157-166, 1994.

[3] Bengio Y.: *Neural Networks for Speech and Sequence Recognition*, Thomson, London, 1995.

[4] Bera A.K., Higgins M.L.: ARCH models: properties, estimation and testing, *Journal of Economic Surveys* **7(4)**, 307-366, 1993.

[5] Bishop C.: *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[6] Bourlard H., Morgan N.: Merging Multilayer Perceptrons and Hidden Markov Models: Some Experiments in Continuous Speech Recognition, Int. Computer Science Institute (ICSI), Berkeley, CA, TR-89-33, 1989.

[7] Box G.E., Jenkins G.M.: *Time Series Analysis*, Holden-Day, San Francisco, 1970.

[8] Broomhead D.S., Lowe D.: Multivariable Functional Interpolation and Adaptive Networks, *Complex Systems* **2**,321-355, 1988.

[9] Chakraborty K., Mehrotra K., Mohan C.K., Ranka S.: Forecasting the Behavior of Multivariate Time Series Using Neural Networks, *Neural Networks* **5(6)**, 961- 970, 1992.

[10] Chappelier J.-C., Grumbach A.: Time in Neural Networks, in *SIGART BULLETIN, ACM Press* **5(3)**, 1994.

[11] Chatfield C.: *The Analysis of Time Series – An Introduction*, Chapman and Hall, London, 4th edition, 1989.

[12] Connor J., Atlas L.E., Martin D.R.: Recurrent Networks and NARMA Modeling, in Moody J.E., et al.(eds.): *Neural Information Processing Systems 4*, Morgan Kaufmann, San Mateo, CA, pp.301-308, 1992.

[13] Cottrell G.W., Munro P.W.: Principal components analysis of images via backpropagation, *Proc.Soc.of Photo-Optical Instr. Eng.*, 1988.

[14] Cybenko G.: Approximation by Superpositions of a Sigmoidal Function, *Math Control Signals Syst*, **2**,303-314, 1989.

[15] Debar H., Dorizzi B.: An Application of a Recurrent Network to an Intrusion Detection System, in *IJCNN International Joint Conference on Neural Networks*, Baltimore, IEEE, pp.478-483, 1992.

[16] DeCruyenaere J.P., Hafez H.M.: A Comparison Between Kalman Filters and Recurrent Neural Networks, in *IJCNN International Joint Conference on Neural Networks*, Baltimore, IEEE, pp.247-251, 1992.

[17] Dorffner G., Leitgeb E., Koller H.: Toward Improving Exercise ECG for Detecting Ischemic Heart Disease with Recurrent and FeedForward Neural Nets, in Vlontzos J., et al.(eds.): *Neural Networks for Signal Processing IV*, IEEE, New York, pp. 499-508, 1994.

[18] Duda R.O., Hart P.E.: *Pattern Classification and Scene Analysis*, John Wiley & Sons, N.Y., 1973.

[19] Elman J.L.: Finding Structure in Time, *Cognitive Science* **14(2)**, 179-212, 1990.

[20] Giles C.L., Miller C.B., Chen D., Chen H.H., Sun G.Z., Lee Y.C.: Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks, *Neural Computation* **4(3)**, 393-405, 1992.

[21] Gordon A., Steele J.P.H., Rossmiller K.: Predicting Trajectories Using Recurrent Neural Networks, in Dagli C.H., et al.(eds.): *Intelligent Engineering Systems through Artificial Neural Networks*, ASME Press, New York, pp.365-370, 1991.

[22] Girosi F., Poggio T.: Networks and the Best Approximation Property, *Biological Cybernetics* **63**,169-176, 1990.

[23] Hertz J.A., Palmer R.G., Krogh A.S.: *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, CA, 1991.

[24] Ho T.T., Ho S.T., Bialasiewicz J.T., Wall E.T.: Stochastic Neural Adaptive Control Using State Space Innovations Model, in *International Joint Conference on Neural Networks*, IEEE, pp.2356-2361, 1991.

[25] Hornik K., Stinchcombe M., White H.: Multi-layer Feedforward Networks are Universal Approximators, *Neural Networks* **2**, 359-366, 1989.

[26] Jordan M.I.: Serial Order: A Parallel Distributed Processing Approach, ICS- UCSD, Report No. 8604, 1986.

[27] Kamijo K., Tanigawa T.: Stock Price Pattern Recognition: A Recurrent Neural Network Approach, in Trippi R.R. & Turban E.(eds.): *Neural Networks in Finance and Investing*, Probus, Chicago, pp.357-370, 1993.

[28] Kolen J.F.: Exploring the Computational Capabilities of Recurrent Neural Networks, Ohio State University, 1994.

[29] Kolen J.F., Pollack J.B.: The Observers' Paradox: Apparent Computational Complexity in Physical Systems, *Journal of Exp. and Theoret. Artificial Intelligence* **7(3)**, 1995.

[30] Kurkova V.: Universal Approximation Using Feedforward Neural Networks with Gaussian Bar Units, in Neumann B.(ed.): *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI92)*, Wiley, Chichester, UK, pp.193- 197, 1992.

[31] Lee C.H., Park K.C.: Prediction of Monthly Transition of the Composition Stock Price Index Using Recurrent Back-propagation, in Aleksander I. & Taylor J.(eds.): *Artificial Neural Networks 2*, North-Holland, Amsterdam, pp.1629- 1632, 1992.

[32] Morgan D.P., Scofield C.L.: *Neural Networks and Speech Processing*, Kluwer Academic Publishers, Boston, 1991.

[33] Mozer M.C.: Neural Net Architectures for Temporal Sequence Processing, Predicting the Future and Understanding the Past, in A. Weigend and N. Gershenfeld (Eds.): *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley Publishing, Redwood City, CA, 1993.

[34] Pollack J.B.: The Induction of Dynamical Recognizers, *Machine Learning* **7(2/3)**, 227-252, 1991.

[35] Port R.F., Cummins F., McAuley J.D.: Naive time, temporal patterns, and human audition, in Port R.F., van Gelder T. (eds.): *Mind as Motion*, MIT Press, Cambridge, MA, 1995.

[36] Refenes A.N., Azema-Barac M., Chen L., Karoussos S.A.: Currency exchange rate predicition and neural network design strategies, *Neural Computing and Applications* **1(1)**, 1991.

[37] Refenes A.N.(ed.): *Neural Networks in the Capital Markets*, Proceedings of the first International Workshop on Neural Networks in the Capital Markets, London, Nov 18-19, 1993.

[38] Refenes A.N., Zapranis A., Francis G.: Stock Performance Modeling Using Neural Networks: A Comparative Study with Regression Models, *Neural Networks* **7(2)**, 375-388, 1994.

[39] Rementeria S., Oyanguren J., Marijuan G.: Electricity Demand Prediction Using Discrete-Time Fully Recurrent Neural Networks, *Proceedings of WCNN'94*, CA, USA, 1994.

[40] Roberts S., Tarassenko L.: The Analysis of the Sleep EEG using a Multilayer Neural Network with Spatial Organisation, *IEE proceedings Part F* **(6)**, 420-425, 1992.

[41] Rohwer R.: The Time Dimension of Neural Network Models, *SIGART BULLETIN, ACM Press* **5(3)**3, 1994.

[42] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning Internal Representations by Error Propagation, in Rumelhart D.E. & McClelland J.L.: *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol 1: Foundations, MIT Press, Cambridge, MA, 1986.

[43] Schmidhuber J.: A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks, *Connection Science* **4(1)**, 403-412, 1989.

[44] Trippi R.R., Turban E.(eds.): *Neural Networks in Finance and Investing*, Probus, Chicago, 1993.

[45] Ulbricht C., Dorffner G., Canu S., Guillemyn D., Marijuan G., Olarte J., Rodriguez C., Martin I.: Mechanisms for Handling Sequences with Neural Networks, in Dagli C.H., et al.(eds.): *Intelligent Engineering Systems through Artificial Neural Networks*, Vol. 2, ASME Press, New York, 1992.

[46] Ulbricht C.: Multi-Recurrent Networks for Traffic Forecasting, in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Cambridge, MA, pp.883-888, 1994.

[47] Ulbricht C.: State Formation in Neural Networks for Handling Temporal Information, Institut fuer Med.Kybernetik u. AI, Univ. Vienna, Dissertation, 1995.

[48] Ulbricht C., Dorffner G., Lee A.: Forecasting fetal heartbeats with neural networks, to appear in: Proceedings of the Engineering Applications of Neural Networks (EANN) Conference, 1996.

[49] Weigend A.S., Rumelhart D.E., Huberman B.A.: Back-Propagation, Weight- Elimination and Time Series Prediction, in Touretzky D.S., et al.(eds.): *Connectionist Models*, Morgan Kaufmann, San Mateo, CA, pp.105-116, 1990.

[50] Weigend A.S., Gershenfeld N.A. (eds.): *Time Series Prediction: Forecasting the Future and Understanding the Past*, Reading, MA: Addison Wesley, 1994.

[51] Vlontzos J., Jenq-Neng H., Wilson E.(eds.): *Neural Networks for Signal Processing IV*, IEEE, New York, 1994.

[52] Waibel A.: Consonant Recognition by Modular Construction of Large Phonemic Time-delay Neural Networks, in Touretzky D.(ed.): *Advances in Neural Information Processing Systems*, Morgan Kaufmann, Los Altos, CA, pp.215-223, 1989.

[53] White H.: Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns, in Trippi R.R. & Turban E.(eds.): *Neural Networks in Finance and Investing*, Probus, Chicago, pp.315-328, 1993.

[54] Widrow B., Stearns S.D.: *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[55] Williams R.J., Zipser D.: Experimental Analysis of the Real-time Recurrent Learning Algorithm, *Connection Science* **1(1)**, 87-111, 1989.

[56] Williams R.J.: Training Recurrent Networks Using the Extended Kalman Filter, in *International Joint Conference on Neural Networks*, Baltimore, IEEE, pp.241-246, 1992.