Neural Computation and Applications in Time Series and Signal Processing

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

Abstract

This paper presents a brief overview on models from neural computation and their applicability to problems in time series and signal processing. Much focus is put on pointing out the relationships between neural networks and more traditional methods for time series analysis. For more details on some of the advanced models, the reader is refered to the bibliography.

1 Neural computation

Neural computation in pattern recognition refers to an array of models and methods originating in the first attempts to formalise information processing in the brain. A typical *neural network*, depicted in figure 1, is of the form

$$x_j^{out} = \sum_{l=1}^{k} v_{lj} f(\sum_{i=1}^{n} w_{il} x_i^{in}) \tag{1}$$

or

$$x_j^{out} = \sum_{l=1}^{k} v_{lj} f(\sqrt{\sum_{i=1}^{n} (w_{il} - x_i^{in})^2}) \tag{2}$$

where x_i^{in} and x_j^{out} stand for input and output values, respectively, and v_{lj} and w_{il} stand for the so-called *weights*, or degrees of freedom, of the models. Equation 1 corresponds to the well-known *multilayer perceptron* (MLP), when f is a so-called *sigmoid function*, such as $f(x) = \frac{1}{1-e^{-x}}$ or $f(x) = \tanh(x)$. Equation 2 corresponds to the *rdial basis function network* (RBFN), with f, for instance, being the Gaussian function $f(x) = \exp(-x^2)$.

The main strength of this type of neural network is that it can approximate any arbitrary nonlinear function $\vec{x}^{out} = F(\vec{x}^{in})$, provided the number

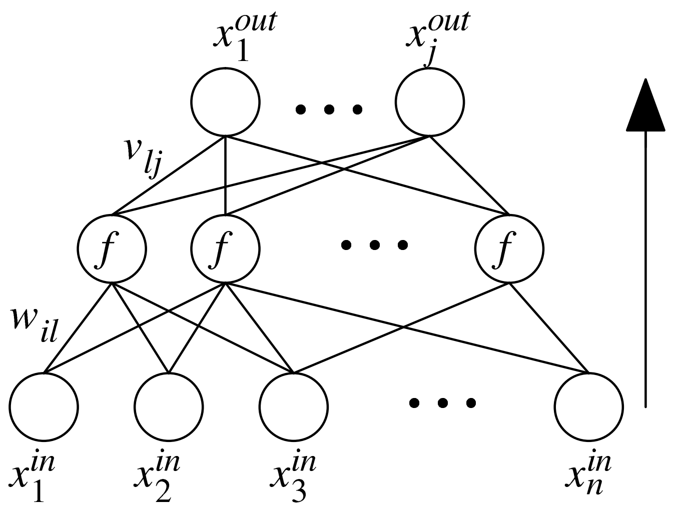


Figure 1: A generic multilayer neural network for function approximation

k of so-called *hidden units* is large enough ([Hornik et al. 1989]). Thus they are called *universal function approximators*. Approximation is done by a weighted superposition of simple nonlinear functions (the sigmoid or the Gaussian). [Bishop 1995] calls this a *semi-parametric* estimation of a function, since on one hand the function F is parameterised through the weights, but within the capacity of a given network little has to be assumed about the shape of the function (similar to non-parametric estimates).

When it comes to pattern recognition, neural networks have little to do with the brain, despite their original motivation (the weighted sum in equation 1 and the sigmoid were introduced to roughly mimic the potential accumulation and firing behavior of biological neurons). Instead, they are advanced methods for nonlinear exploratory and inductive statistics. *Learning* in neural networks (also called training) must be seen in the same realm. Weights are usually derived during model estimation in a *maximum likelihood* framework, using data from a so-called training set. In its simplest form (see below), maximizing likelihood amounts to minimizing the *summed squared error* $E = \sum_{i=1}^{N}(x_i^{\text{target}} - x_i^{\text{out}})^2$, where x^{target} is given by the training samples. Two types of applications are usually distinguished: *regression* – i.e. estimating continuous output values – and *classification*. For more details on neural networks for pattern recognition, see [Bishop 1995].

Neural computation nowadays encompasses a much wider array of methods than the original neural networks. They include *support vector machines* (for which the lower part of the network is kept fixed through the choice of proper *kernel functions*), *independent component analysis* for source separation, *Gaussian* and other *mixture models*, and many types of unsupervised learning methods. What they all have in common is that

- there is a strong focus on nonlinearity
- complexity is approximated by superpositions of simpler building blocks
- thus, focus is on semi-parametric methods

In this paper, we restrict ourselves to universal approximators (such as the MLP or RBFN), since they are mostly used in time series and signal processing. An important extension, which is of interest with respect to time series, are so-called *recurrent* neural networks. They are characterized by the introduction of feedback connections from hidden or output units to the input layer (see figure 7 and 8 below).

2 Time Series Processing

Time series processing is the field of pattern recognition and analysis of time-varying data. A typical time series problem is given by a vector of observable measurements at consecutive points in time t:

$$\vec{x}_t, t = 0, 1, ... \tag{3}$$

If $\vec{x}$ is indeed a vector, i.e. more than one variable is observed, one speaks of a *multivariate time series*: if it is a scalar, the time series is called *univariate*. The representation of a time-varying set of variables in equation 3 makes several important assumptions:

- Time is discrete, meaning that the observables are measured only at discrete points in time. This is different from so-called *continuous time* models. In signal processing terms one could speak of "sampling" the original process observable into the variable vector $\vec{x}_t$.
- Points in time for measurement are equi-distant, meaning that there is a constant time interval between points of measurement.

We restrict our discussion in this paper to this kind of discrete-time processes, since they are the most common models for practical applications and are most amenable to analysis with common methods from neural computation.

The length of the time interval depends on the type of application. Typical examples of time series are

- sampled acoustic or biosignals (time interval usually milli- or nanoseconds)

- measurments of oxygen saturation in intensive care (typically seconds)
- measurements of process parameters in industry (typically seconds or minutes)
- temperature measurements in meteorology (typically hours)
- stock or option prices in the financial markets (typically days)
- econometric measures like inflation rate (typically weeks)
- the number of sunspots (years)

This overview of possible applications highlights that there is no principle difference between what is called *signal processing* and what is called *time series processing*. Formally the methods are the same or very similar, only the time interval (short for signals, longer for time series) and the focus of the application usually differs. In signal processing, typical problems are

- Filtering of the signal, i.e. changing its general characteristics
- Source separation, i.e. considering the signal as a mixture of unknown sources and dividing it into them

Typical problems in the domain of time series processing are

- Forecasting, i.e. estimating the future development of the time series
- Noise modeling, i.e. estimating the stochastic variability of the time series

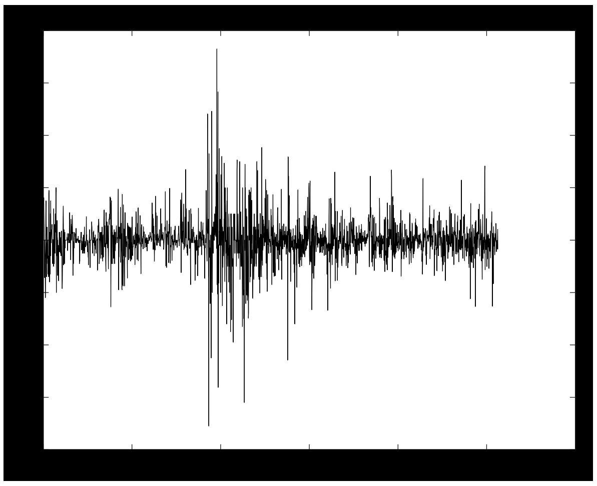
Problems common to both domains are

- Pattern recognition, i.e. recognising typical wave forms or subsequences
- Modeling of the underlying process, i.e. finding a mathematical model that describes the generating process underlying the observable variables.

In this paper, the focus will be on modeling and forecasting.

Figure 2 depicts two typical time series, which will be used as examples in this paper – one from finance, namely returns from the daily Austrian stock exchange index ATX, and one from astronomy, namely the annual number of observed sunspots since the 18th century (this is a well-known benchmark time series).

The former is a typical case of a rather noisy time series. It is derived from differencing the original time series of daily index values by calculating



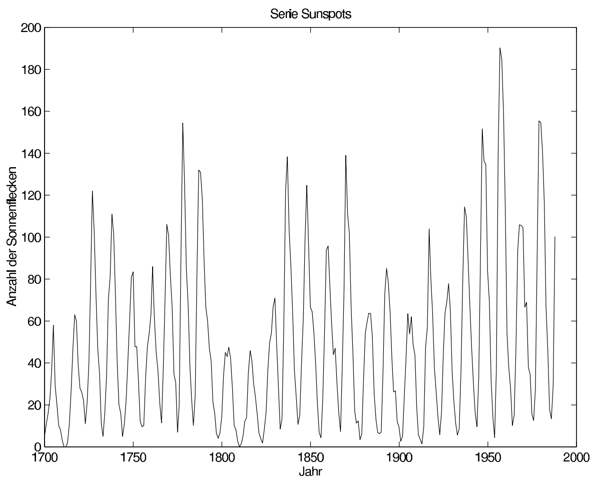


Figure 2: Two typical time series: Returns from the Austrian stock exchange index ATX (left), and the annual number of subspots since 1770 (right).

$$r_t = x_t - x_{t-1} \tag{4}$$

Differencing is a proper preprocessing method for many applications in order to remove *trends* and some *instationarities*. In financial applications this has the direct interpretation as “returns”, i.e. potential wins or losses one would get when trading the commodity.

The latter is a typical case of a more structured time series. What can be observed are distinct so-called *seasonalities*, i.e. recurring periodic patterns in the time series. For optimal modeling, such seasonalities should also be removed, e.g. by the following type of differencing:

$$y_t = x_t - x_{t-s} \tag{5}$$

where s is the time interval between seasonal peaks.

3 Forecasting as modeling

In this section we will see that forecasting time series is akin to finding a model for the generating process. We will restrict ourselves to univariate time series, keeping in mind that all models can easily be extended to the mulitvariate case.

3.1 Autoregressive modeling and feedforward networks

Forecasting can be interpreted as making optimal use of past information to predict the future. Of course, in real-world applications most of the time we must assume stochasticity, i.e. forecasting can only lead to an estimate in terms of an *expected value* or *expectation*, from which actual observations will differ due to unpredictable influences, modeled as a noise process.

One of the most common assumptions in time series forecasting is based on taking past observations as the sole past information available. The expected value is assumed to a be a function of a fixed, and limited, number of past observations. The noise process is assumed to be additive. This leads to the following expression:

$$x_t = F(x_{t-1}, x_{t-2}, \ldots, x_{t-p}) + \epsilon_t \tag{6}$$

For convenience we denote $X_{t,p} = (x_{t-1}, x_{t-2}, \ldots, x_{t-p})$. ϵ_t is usually refered to as a *random shock*. This type of model is generally called an *autoregressive* (or AR) model, since it amounts to a general regression of the observable variable x_t over its own past values. p is called the *order* of the model and corresponds to the number of past observations used in the regression. The best forecast after model estimation is to output the expected value $\mu_t = F(X_{t,p})$.

Using a limited number of past observations corresponds to the common *Markov assumption*, which states that all we have to know to predict the future is the present state of the system, in this case given by the vector of p past observations. In other words, we assume that we do not have to know the entire evolution of the time series in order to predict.

It is obvious that equations like 6 can be seen both as a model for forecasting (i.e. the best prediction is, after an appropriate estimation of the parameters describing f, to forecast the expected value $\mu_t = \hat{x}_t = F(X_{t,p})$) and as a generative model describing the underlying process. Given proper starting values, equation 6 can be used to generate time series values with the same characteristics as the original observations, drawing ϵ_t from the assumed distribution.

The most commonly used version in time series processing literature is the class of *linear* AR models, i.e. where $F(x)$ is a linear function of the following type:

$$x_t = \sum_{i=1}^{p} a_i x_{t-i} + \epsilon_t \tag{7}$$

The noise process behind ϵ_t is usually assumed to be *identically independently distributed* (i.i.d.), typically following a Gaussian distribution with zero mean and a given variance σ^2, i.e. $\epsilon \sim N(0, \sigma^2)$.

The simplest form of a linear AR process is one of order 1 with $a_1 = 1$ and $\epsilon \sim N(0, 1)$:

$$x_t = x_{t-i} + \epsilon_t \tag{8}$$

This process is called *random walk*. The expected value, and therefore the best possible prediction, for x_t is the previous value x_{t-1}. In other words, the process is characterised by the fact that at each time step an i.i.d. disturbance ϵ_t is added to the observed variable. If a given time series follows a random walk, the best possible prediction is trivial and leads to no new information. This process is especially important for domains like financial time series processing, since it corresponds to the hypothesis of an efficient financial market. The hypothesis says that at each time step (e.g. each day) all available information that could be used to beat the market (i.e. to profit from a non-trivial prediction) has already been absorbed by the market mechanisms, rendering such profitable forecasting impossible. But also in other domains it is important to keep the random walk in mind before applying any more complex prediction method, and always benchmark such a model against it.

Naturally, more complex time series processes exist, also in the financial markets, and thus its is now worthwhile to explore how neural computation can enhance the common linear AR models. From equation 6 the main contribution from neural computation in this context becomes clear: any universal approximator can potentially be used to model an arbitrary nonlinear function $F(x)$. An example is shown in figure 3. A multilayer perceptron, as an example, can be used to model an arbitrary *nonlinear autoregressive process*. The way the past p observations x_{t-i} are used is usually called *time window*, or sometimes a special form of *time delay*, in neural network literature.

3.2 Complex noise models

Nonlinearity is not the only sensible extension to the classical linear AR model in equation 7. Recently, research has focused on modeling more complex noise processes than the ususal Gaussian distribution with constant variance σ^2 ([Schittenkopf et al. 2000, Neuneier et al. 1994, Husmeier 1999]).

[Bishop 1995] has demonstrated that the minimization of the summed squared error in regression (and, therefore, in autoregression), assuming linear output activation functions, corresponds to a maximum likelihood estimation assuming constant Gaussian noise $N(0, \sigma^2)$. After estimation ("learning"), σ^2 corresponds to the normalised residual quadratic error. This is illustrated in figure 4. An autoregressive process of order 1 can be visualised by plotting all past observations x_{t-1} against the present observations x_t, to be forecast. The noise process determines how the actual observations

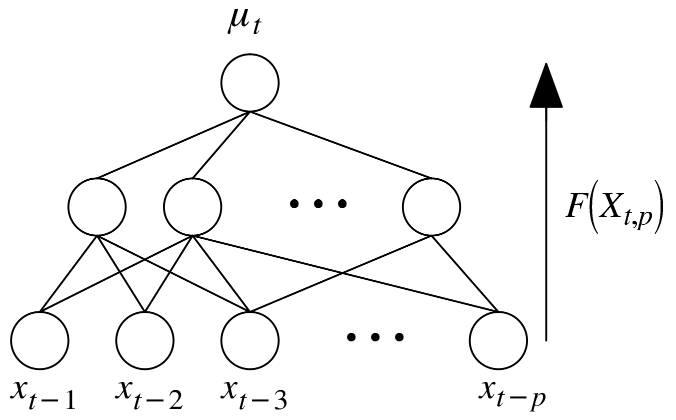


Figure 3: A feedforward neural network as a nonlinear autoregressive model.

are distributed around the expected value, given by the, potentially nonlinear, function $F(x_{t-1})$.

The illustration in figure 4 makes clear that the noise around the expected value does not have to be constant, nor does it have to be Gaussian. Instead, it can be seen as also being dependent on the input, i.e. the past observations. In formal terms

$$\epsilon \sim D(\vec{\theta}), \tag{9}$$

where

$$\vec{\theta} = g(X_{t,p}) \tag{10}$$

D is an arbitrary parameterised distribution with parameters $\vec{\theta}$ and is called the *conditional distribution*, since it depends on the past. One could also speak of a time-dependent noise distribution, since it is permitted to change at every time step.

Estimation of the models is straight-forward if one sticks to the maximum likelihood framework suggested above. The model likelihood becomes

$$L = \prod_{i=1}^{N} d(\vec{\theta} = g(X_{t,p}^{(i)})) \tag{11}$$

where N is the number of time series samples in the training set, and d is the probability density function corresponding to D.

The simplest extension to the standard AR case is sticking to the Gaussian distribution but keeping the variance σ^2 time-dependent. The likelihood in this case is

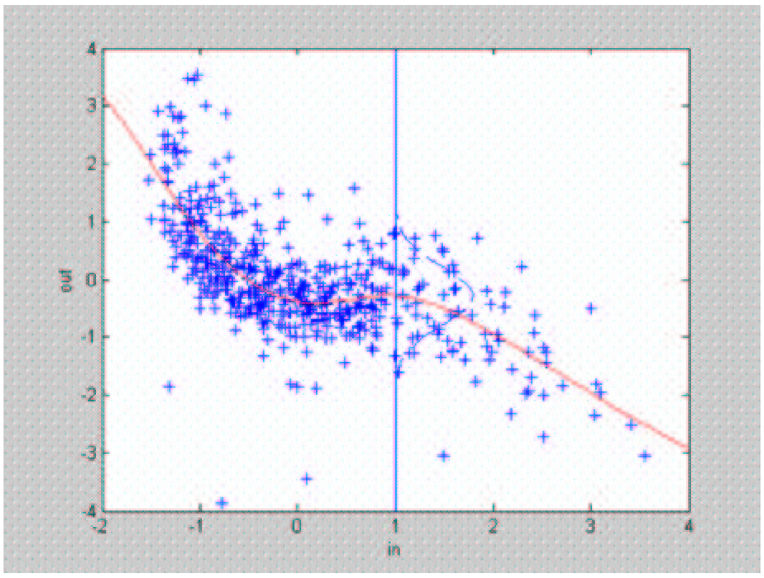


Figure 4: Plotting x_t over x_{t-1} can visualize the role of the noise process in autoregression. For any particular input x_{t-1}, the density function of the noise process ϵ_t describes the distribution of actual observations corresponding to that input. This makes clear that the noise does not have to constant, i.e. independent from x_{t-1}, or even Gaussian.

$$L = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2(X_{t,p})}} e^{-\frac{(x^{(i)}-\mu(X_{t,p}))^2}{2\sigma^2(X_{t,p})}} \tag{12}$$

In neural networks terms this means an additional output unit corresponding to the estimate of σ^2, as depicted in figure 5. In time series literature this case is known as a *heteroskedastic* time series, i.e. a time series with time-dependent variance of its noise process. This is of particular interest in financial time series analysis, since it corresponds to the case of a time-dependent *volatility* of a commodity. Looking at the ATX return series, one sees that this is apparently the case. At different points in time the variance of returns appears to be of different quantity, which can be modeled by a heteroskedastic process. When building a model for option pricing, this dependency is of particular interest. But this is not the only application where heterosketasticity plays an important role. The fact that knowing about the variance of the noise process permits the estimation of a reliable confidence interval in which actual observations are expected to lie highlights the great potential of using such extended models. The noise process in this case is no longer i.i.d. – the distributions guiding the random shocks are still independent, but no longer identical (although still all Gaussian).

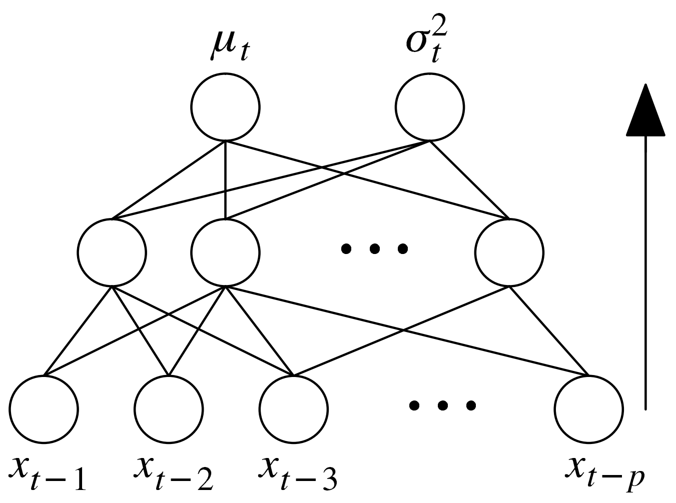


Figure 5: A simple extension to the feedforward neural network to account for heteroskedasticity. An additional output for σ_t^2 is added to account for its potential nonlinear dependence on $X_{t,p}$

What has been said about the general (nonlinear) neural network case is, of course, also true for the linear case. It is worth noting that a particular type of linear[1] case to model heteroskedastic time series is the well-known *autoregressive conditional hetereoskedasticity* (ARCH) model [Engle 1982], frequently used in finance. The ARCH model is defined as

$$\sigma_t^2 = \sum_{i=1}^{p} a_i r_{t-i}^2 \tag{13}$$

where r_{t-i}^2 are past returns of the time series (i.e. the values of the differenced original time series) and σ_t^2 is the variance of a Gaussian distribution. In this case, μ_t, i.e. the expected value of the time series r_t, is assumed to be 0 (or modeled separately using an AR model, replacing r_t by the residuals of that process). Comparing equation 13 to 12 one sees that the neural network case is a nonlinear generalisation of the ARCH model.

Another extension is to choose a parametric probability density function other than the Gaussian, in order to model specific characteristics of the conditional distribution of time series values. In financial time series analysis, it is known that conditional distributions (as well as unconditional ones) can have a higher kurtosis than a Gaussian. For these purposes, the use of a student t-distribution is rather commen.

[1]Strictly speaking, the ARCH model is quadratic in r_{t-i} but linear in r_{t-i}^2

In the spirit of neural computation, an even more general model appears appropriate in those cases where conditional distributions are unknown but expected to be non-Gaussian. Any arbitrary density function can be approximated by a *mixture of Gaussians.* Consequently, [Bishop 1994] has suggested the so-called *mixture density network* (MDN) to model arbitrary noise distributions in regression. Several authors ([Schittenkopf et al. 2000, Neuneier et al. 1994, Miazhynskaia et al. 2003]) have demonstrated the viability to use mixture density networks for arbitrary conditional return distributions in financial time series analysis.

The definition of a mixture density network is straight-forward given equation 11, if one inserts the following density function:

$$d(\vec{\mu}, \vec{\sigma^2}, \vec{\pi}) = \sum_{i=1}^{k} \frac{\pi_i(X_{t,p})}{\sqrt{2\pi\sigma_i^2(X_{t,p})}} e^{-\frac{(x-\mu_i(X_{t,p}))^2}{2\sigma^2(X_{t,p})}} \tag{14}$$

Introducing arbitrary nonlinearity, this amounts to a neural network with $3k$ outputs, each one corresponding to one of the parameters of the mixture, in a straight-forward extension of the network in figure 5. Since there is no reason to assume that the three sets of paramaters – namely the centers μ_i, the widths σ_i^2 and the weights, or priors, π_i – have related nonlinear dependencies on the past, the choice of three different networks is often more appropriate. Learning (model estimation), as was the case for the simpler cases above, amounts to maximising the corresponding likelihood function (minimising the negative log likelihood, respectively). The corresponding error, or loss, function no longer corresponds to a simple squared error, however.

The advantage of using mixture density networks in forecasting is that now arbitrary conditional distributions can be modeled in a semi-parametric way. By replacing the multilayer network with a single-layer perceptron, linear versions can be obtained as well. [Miazhynskaia et al. 2003] has shown that this leads to more reliable confidence intervals for the estimated forecasts, which in their case can be used in risk analysis.

One remark, however, is in place. Mixture models – similar to the approximation of nonlinear functions in a neural network – usually suffer from an *identifiability* problem. In other words, parameters resulting from estimations cannot be interpreted by assigning a meaning to them. Similarly, it can no longer be guaranteed that it is really "pure" noise that is modeled. This is exemplified in figure 6, depicting an MDN estimate for the sunspot time series[2]. An autoregressive mixture density network of order 1 is used to model the distributions x_t (plotted against x_{t-1}). It is obvious that an AR model of order 1 is insufficient to model the structure behind the time series.

[2]Note that the original, although normalised, time series is used, without removing seasonalities

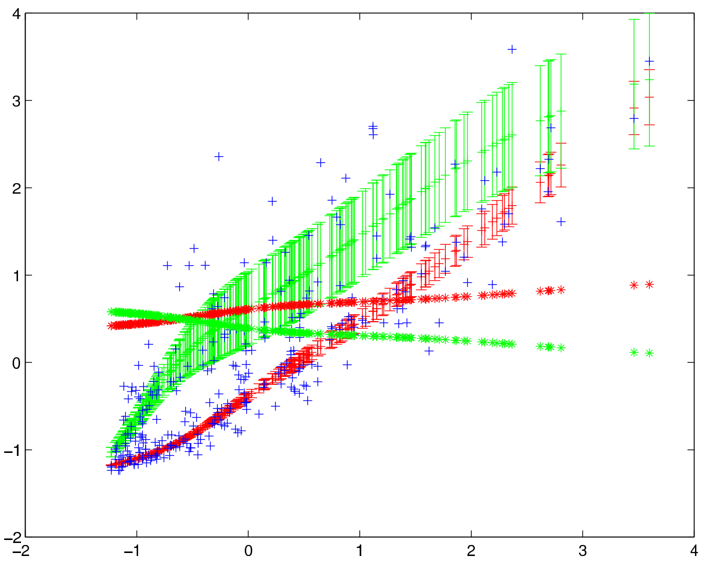


Figure 6: The resulting estimations from training a mixture density network with the subspot data. Two lines of error bars are drawn depicting the centers and widths of the two Gaussians of the mixture, dependent on the input x_{t-1}. Especially in the middle range, the resulting mixture density is bimodal reflecting the fact that in an AR(1) view, for each input the next value is about as likely to be higher as it is to be lower. The two additional lines depict the priors π_i for each Gaussian in the mixture.

Given a time series value around 0, the probability of the series to go up is about equal to it going down. For reliable forecasts of the expected value, an order of at least 2 would be necessary. The resulting model estimate in figure 6 describes this as a bimodal distribution for each x_{t-1} around the value 0. This apparently complex noise process thus captures some of the structure in the data, which would more appropriately be captured by a second-order AR process. Choosing too general a model, therefore, no longer permits the strict distinction between structure in the data and noise. Therefore such models should be used with care.

3.3 Moving average models and recurrent networks

Another common class of models uses a different kind of past information for forecasting. Instead of past time series values, the expectation for x_t is assumed to depend on past random shocks ϵ_{t-i}. In the linear case this amounts to:

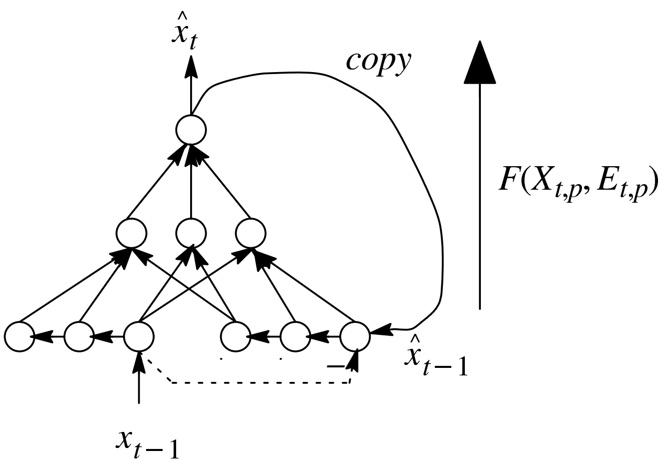


Figure 7: A recurrent Jordan network as an instantiation of a nonlinear moving average model.

$$x_t = -\sum_{i=0}^{q} b_i \epsilon_{t-i} \tag{15}$$

For convenience, we denote $E_{t,q} = (\epsilon_{t-1}, \epsilon_{t-2}, \ldots \epsilon_{t-q})$. By letting the index run from 0 to q, the current random shock is included in this model ($b_0 = 1$). This type of model is called a *moving average* (MA) process of order q. It has been shown that any finite AR process corresponds to an MA process of infinite order, and vice versa. Still, a finite MA process, or even the combination of AR and MA processes (so-called ARMA models) can be viable models for a given time series.

In analogy to above, neural networks can be used to generalise linear MA process to arbitrary nonlinear versions. The question is, however, what input to use to the network, if past random shocks are not really known. [Connor et al. 1992] and [Dorffner 1996] have demonstrated that using past estimates, i.e. past outputs of the network, as inputs, in the limit of the model converging to the true model, amounts to being equivalent to using past random shocks as inputs. If the network outputs the correct expected value for x_t, $\hat{x}_t = \mu_t$, then $\epsilon_{t-1} = x_{t-1} - \hat{x}_{t-1}$. In other words, $\hat{x}_{t-1}$ implicitly contains the information about the past random shock and can thus be used as an input to the network. Using past estimates amounts to a recurrent connection of the network's output to the input (see figure 7), i.e. a recurrent network usually called a *Jordan network* (see [Dorffner 1996] for the theoretical equivalence to an MA process in the limit).

We can therefore conclude: Recurrent Jordan networks are instantiations of nonlinear moving average processes. Little is known, however, about convergence properties of this type of model (i.e. whether and how quickly they converge toward the true model, which is a prerequisite for being an MA process).

In this context, it is worth looking at another common time series model and its nonlinear generalisation. For heteroskedastic time series, a property known as *volatility clustering* is often observed. This means that a high variance (high volatility) is often followed by several time steps of high variance. For instance, in the financial markets large shocks tend to prevail for some time. To model this property, the GARCH model (generalised autoregressive conditional heteroskedasticity) was introduced ([Bollerslev 1986]):

$$\sigma_t^2 = \sum_{i=1}^{p} a_i r_{t-i}^2 + \sum_{i=1}^{p} b_i \sigma_{t-i}^2 \tag{16}$$

Couched in neural networks terms, as the nonlinear ARCH above, this again means using past estimates (this time of σ^2) as input to the network. Therefore, a nonlinear GARCH model can be obtained by introducing the *recurrent mixture density network* [Schittenkopf et al. 2000, Tino et al. 2001].

3.4 State space models and recurrent networks

Another common method for time series processing are so-called *state space models* [Chatfield 1989]. Here the assumption is that the current state (in terms of the Markov assumption) is not given by the past observations directly but is hidden. The observations thus depend on a state vector $\vec{s}$:

$$x_t = \mathbf{C}\vec{s}_t + \epsilon_t \tag{17}$$

where $\mathbf{C}$ is a transformation matrix. The time-dependent state vector is usually modeled by a (multivariate) linear AR(1) process:

$$\vec{s}_t = \mathbf{A}\vec{s}_{t-1} + \mathbf{B}\vec{\eta}_t \tag{18}$$

where $\mathbf{A}$ and $\mathbf{B}$ are matrices, and $\vec{\eta}_t$ is a vectorial noise process, similar to ϵ_t above.

If we further assume that the states are also dependent on the past time series observations (an assumption, which is common, for instance, in signal processing – see [Ho et al. 1991]), and neglect the additional noise term $\mathbf{B}\vec{\eta}_t$:

$$\vec{s}_t = \mathbf{A}\vec{s}_{t-1} + \mathbf{D}X_{t,p} \tag{19}$$

then we basically obtain an equation describing a recurrent neural network type, known as *Elman network* (after [Elman 1990]), depicted in figure 8. 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,

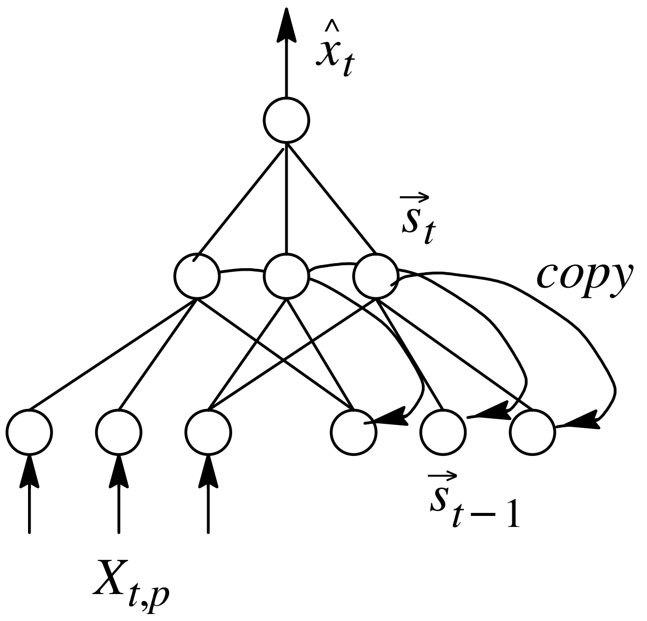


Figure 8: The Elman recurrent network as an instantiation of the state-space model.

and equate the activation vector of the hidden layer with $\vec{s}$, the only difference to equation 19 is the fact that in an MLP a sigmoid activation function is applied to the input of each hidden unit:

$$\vec{s}_t = f(\mathbf{A}\vec{s}_{t-1} + \mathbf{D}X_{t,p}) \tag{20}$$

where f is a sigmoid function. 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).

Like above, the strong relationship to classical time series processing can be exploited to introduce "new" learning algorithms. For instance, in [Williams 1992] the *Kalman algorithm*, developed for the original state space model (Kalman filter) is applied to general recurrent neural networks.

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 17 and 18 by an arbitrary non-linear function, one obtains

$$\vec{x}_t = F_1(\vec{s}_t) + \vec{\epsilon}_t \tag{21}$$

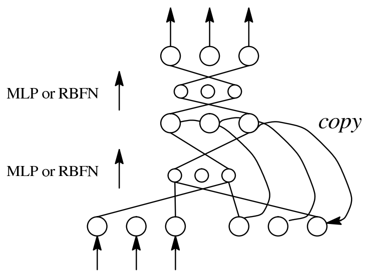


Figure 9: An extension of the "Elman" network as realization of a non-linear state-space model

$$\vec{s}_t = F_2(\vec{s}_{t-1}) \tag{22}$$

Like in the previous sections on non-linear ARMA models, these non-linear functions F_1 and F_2 could be modeled by an MLP or RBFN. The resulting network is depicted in figure 9. An example of the application of such a network is [Kamiho et al. 1993].

It has frequently been noted in literature that the state vector $\vec{s}$ apparently implicitly contains information of the *entire* past of the time series. Therefore, recurrent networks seemingly are not limited to a fixed time horizon, as are ARMA models. In practice, however, this is not true. The influence of past information vanishes exponentially, leading to a model that actually only takes recent information into account. A similar observation applies to model estimation. The gradient in minimizing the negative log likelihood, which must be "propagated back" via the recurrent connections, also vanishes exponentially ([Bengio et al. 1994]), rendering training difficult in many cases. Thus there is evidence that in practical applications, recurrent networks can have a disadvantage against feedforward networks, although potentially equally powerful (see, for instance, [Hallas & Dorffner 1998]).

4 Discrete valued and symbolic time series

So far, we have discussed time series with continous values x_t. In many applications, however, observations can only take one of a small finite set of values. Examples are binary measurements or very coarsely quantised signal values. A special case are *symbolic time series*, where values do not have an order (or the order is neglected) and can be considered as symbols from a given finite alphabet, i.e. $x_t \in \{s_i\}$, where s_i are arbitrary symbols. Examples are letters in a text, amino acids in a gene string, or continuous time series that have been quantised into a small number of intervals (e.g. 'up' and 'down' for stock price returns).

Forecasting can again be viewed as estimating expected values based on past information. The main difference is that probability densities are replaced by discrete probabilities for the symbols in the alphabet. The equivalent to an AR model would be looking at the string of past p symbols and finding estimates for each symbol to follow the string. This conditional probability distribution can be denoted as $P(x_t|x_{t-1}x_{t-2}\dots x_{t-p})$, where the condition part denotes the concatenation of the past p symbols.

This type of model is called a *Markov chain* (or Markov model) of order p. The probabilities can easily be estimated as the empirical probabilities (frequencies) that each symbol in the alphabet follows the particular given string. However, this type of model easily runs into problems for higher orders, since long substrings can be rather rare in the entire time series, rendering the estimation of empirical probabilities impossible.

Therefore, several authors ([Ron et al. 1996, Tino & Dorffner 2001]) have proposed so-called *variable-length Markov models* which only consider contexts (i.e. substrings) that occur frequently enough in the time series. The approach by [Tino & Dorffner 2001], the so-called *fractal prediction machine* (FPM), not only has a very intuitive geometric interpretation, but also bears an important similarity to another class of recurrent neural network. Basically, each potential substring is mapped onto a point in a $\log_2(n)$-dimensional space (where n is the size of the alphabet), following a simple mapping borrowed from iterated function systems ([Barnsley 1988]). This is illustrated in figure 10. When this is done for every substring of length p, the resulting distribution of points has the following interesting property: Strings that have a large common suffix (i.e. have a large common ending substring) are mapped onto points that are close together in space. Therefore, clusters of points correspond to substrings that frequently occur in the time series. Using a simple clustering algorithm, this can be used for identifying substrings, for which a reliable probability estimate for subsequent symbols can be found.

The result can be viewed as a stochastic automaton, each state of which forms an equivalence class of substrings. For each such state, probabilities can be estimated by calculating the relative frequencies of symbols that follow. This view leads to an analogy to state space models and recurrent neural networks, as was exemplified above for continuous-valued time series. As shown in [Tino & Dorffner 1998], the mapping of the FPM corresponds to the recurrent part of a second-order neural network, while the feedforward part would take the role of estimating probabilities for each state.

5 Signal filtering

As stated above, there is no principled formal distinction between what is usually termed *time series processing* and *signal processing*. The main differ-

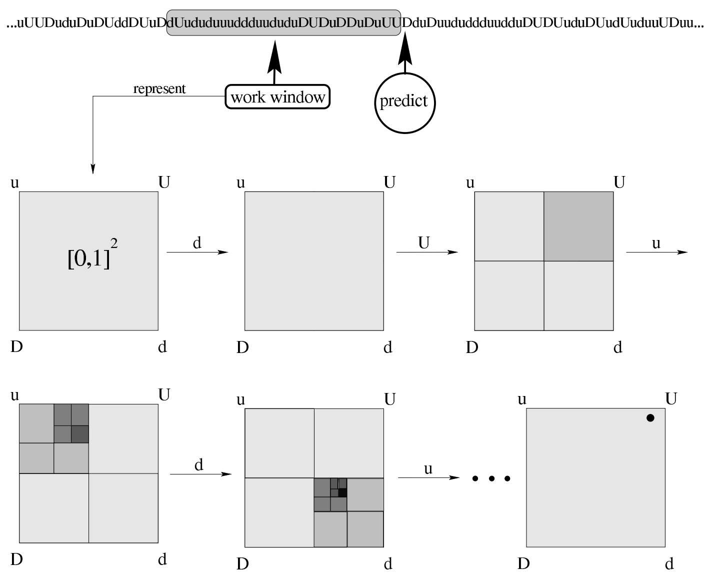

...uUUDuduDuDUddDUuDdUududuuudduududuDUDuDDuDuUUDduDuududdduudduDUDUuduDUudUuduuUDuu...
represent
work window
predict
u
U
[0,1]^2
D
d
d
U
u
...

Figure 10: The basic mechanism of a fractal prediction machine to map substrings of symbolic sequences onto points in space: Each symbol in the alphabet is assigned a corner in the $\log_2(n)$-dimensional space for an alphabet of size n. For a given work window, all symbols in the resulting substring are processed in order, starting with the most recent one. As in an iterative function system, an affine mapping of the entire space onto one of the n subspaces ("corners") is performed, one particular mapping for each symbol. By tracing the center point of the space through all the mappings, the end point reached corresponds to the entire substring. The interesting property is that substrings with long common suffixes are mapped to points that are close in space.

ence lies in how problems are phrased and not in the chosen methodology. A common problem for signal processing is finding appropriate *filters* to describe or generate signals.

Digital filters can be depicted in a way that is pratically identical to autoregressive and moving average models. The former case, for instance, corresponds to the so-called *finite impulse response filters*. Therefore, there is a strong correspondance between forecasting and filtering models.

Autoregressive models can also be shown to be equivalent to the well known *spectral analysis* of signals. In other words, parameters of a linear AR model of a given order can be used as signal descriptors in a similar way as a spectrum derived from Fourier analysis. This gives rise to applications in recognizing typical signal patterns (waveforms) or to classify signals. A typical example is the classification of electroencephalographic (EEG) recordings of a person during sleep into one of several sleep stages (see, e.g., [Sykacek et al. 2002]).

Another important function of AR filtering is that of *denoising*. According to the formulation in equation 6, the model divides the signal into signal content (the predictable component) and a (usually white) noise process. The residuals after model estimation thus corresponds to noise that can be removed from the original signal.

From what we have seen in time series processing, it becomes clear that neural networks lend themselves for nonlinear extensions of classical linear filters ([Haykin 1986]). However, they should be seen as means to an end (i.e. more optimal denoising) but due to the identifiability problem mentioned above they cannot be used in the same way to describe signal characteristics in a parametric manner.

6 Practical considerations

We have seen that neural networks are powerful models that can be used in various time series and signal processing applications. Straight-forward extensions of simple mathematical principles have lead to advanced models desribing time series and signals in an intricate way (e.g. with respect to the noise process). Power in modeling, however, always comes with a price that has to be paid through extra care and sound validation techniques, without which neural networks are easily mis-applied.

In particular,

- Semiparametric nonlinear techniques need a sensible model selection and validation strategy. In general pattern recognition, usually resampling strategies such as n-fold cross-validation are applied to this avail – meaning that multiple runs with different training and independent validation sets must be performed. In time series processing it can be shown that, in order to be truly independent, validation sets should always be observations that occur *after* the training set. This leads to a sliding window technique that should be applied when validating process models in a maximum likelihood framework (see, for instance, [Schittenkopf et al. 2000]).

- Models with a large number of degrees of freedom (e.g. weights in a neural network) need large number of training samples. Stationarity of the time series becomes an important issue here. In other words, training sets for time series processing can often not be extended to arbitrary size, since the main characteristics of the time series or signal might change. Thus there often is an inherent limit to the complexity of the models that can reliably be estimated.

- It is not a priori clear for a given time series whether nonlinearity indeed plays a large role. In the financial markets, for instance, there is growing evidence that arbitrary nonlinearity in a model cannot significantly improve forecasting performance. Therefore, any complex neural network model should always be carefully validated against its linear or otherwise more parametric counterparts.

- Identifiability, as mentioned several times in this paper, might not be a problem for many engineering solutions (e.g. finding good forecasts). But for many applications (e.g. filtering) it does play a role and often leaves neural networks useless (or at least, difficult to deal with), despite their potential power in modeling.

7 Summary and conclusions

The purpose of this paper was to give a short overview of the potential of neural computation methods in modeling time-varying data. Much emphasis was put on showing that neural networks are embedded in more traditional time series theory and have the potential to provide powerful alternatives and extensions, especially with respect to nonlinearity. Time series and signal processing, however, is a field with a long traditon that has not waited for neural computation to provide viable models. Linear ARMA models, Markov chains, linear filters, etc. are rather powerful in themselves and must therefore always be considered before blindly applying a neural network.

Many important topics have not been addressed, such as deterministic chaos in nonlinear processes, the relationship between recurrent networks and stochastic automata, etc. Also, a large part of neural computation, from support vector machines, wavelet networks to independent component analysis could not be dealt with. Nevertheless, the hope is that the reader could get a glimpse of the fascinating potentials of advanced models to desribe time-varying data, which is prevalent in a great many of practical applications.

References

[Barnsley 1988] Barnsley, M.F.: *Fractals everywhere*, Academic Press, New York, 1988.

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

[Bishop 1994] Bishop, C. M.: Mixture density networks, Neural Computing Research Group Report: NCRG/94/004, Aston University, Birmingham, 1994.

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

[Bollerslev 1986] Bollerslev, T.: A generalized autoregressive conditional heteroskedasticity, *Journal of Econometrics*, **31** (1986), 307-327.

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

[Connor et al. 1992] 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.

[Dorffner 1996] Dorffner G.: Neural networks for time series processing, Neural Network World, 6(4)447-468, 1996.

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

[Engle 1982] Engle, R. F.: Autoregressive conditional heteroskedasticity with estimates of the variance of U.K. inflation, *Econometrica*, **50** (1982), 987-1008.

[Hallas & Dorffner 1998] Hallas M., Dorffner G.: A Comparative Study on Feedforward and Recurrent Neural Networks in Time Series Prediction Using Gradient Descent Learning, in Trappl R. (ed.), Cybernetics and Systems '98 - Proc. of 14th European Meeting on Cybernetics and Systems Research, Austrian Society for Cybernetic Studies, Vienna, pp.644-647, 1998.

[Haykin 1986] Haykin S.: Adaptive Filter Theory, Prentice-Hall, London/New York/Englewood Cliffs, NJ, 1986.

[Ho et al. 1991] 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.

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

[Husmeier 1999] Husmeier D.: Neural Networks for Conditional Probability Estimation, Springer, Berlin/Heidelberg/New York/Tokyo, 1999.

[Kamiho et al. 1993] 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.

[Miazhynskaia et al. 2003] Miazhynskaia T., Dorffner G., Dockner E.: Risk Management Application of the Recurrent Mixture Density Network Models, to appear in: *Proceedings of ICONIP/ICANN 2003*, Springer Verlag, 2003.

[Neuneier et al. 1994] Neuneier R., Finnoff W., Hergert F., Ormoneit D.: Estimation of conditional densities: a comparison of neural network approaches", in: Marinaro M., and P. G. Morasso (eds.) *ICANN 94 - Proceedings of the International Conference on Artificial Neural Networks*, Berlin: Springer, 689-692.

[Ron et al. 1996] Ron, D., Singer, Y., Tishby, N.: The power of amnesia. *Machine Learning* 25, 1996.

[Schittenkopf et al. 2000] Schittenkopf C., Dorffner G., Dockner E.J.: Forecasting time-dependent conditional densities: A semi-nonparametric neural network approach, Journal of Forecasting, 19, 355-374, 2000.

[Sykacek et al. 2002] Sykacek P., Dorffner G., Rappelsberger P., Zeitlhofer J.: Improving biosignal processing through modeling uncertainty: Bayes vs. non-Bayes in sleep staging, Applied Artificial Intelligence, 16(5)395-421, 2002.

[Tino & Dorffner 1998] Tino P., Dorffner G.: Recurrent Neural Networks with Iterated Function Systems Dynamics, in NC'98, Proceedings of the ICSC/IFAC Symposium on Neural Computation, Vienna, Austria., pp.526-532, 1998.

[Tino & Dorffner 2001] Tino P., Dorffner G.: Predicting the future of discrete sequences from fractal representations of the past, Machine Learning, 45(2)187-217, 2001.

[Tino et al. 2001] Tino P., Schittenkopf C., Dorffner G.: Financial volatility trading using recurrent neural networks, IEEE Transactions on Neural Networks, 12(4)865-874, 2001.

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