A neural network applied to economic time series

by

Gerrit Draisma[†], Johan F. Kaashoek[†] and Herman K. van Dijk[‡]

[†]Econometric Institute, Erasmus University Rotterdam
P.O.Box 1738, 3000 DR Rotterdam, The Netherlands

[‡]Tinbergen Institute Rotterdam
Oostmaaslaan 950-952, 3063 DM Rotterdam, The Netherlands

Abstract

A major problem in applying neural networks is the determination of the size of the network. Even for moderate networks the number of parameters can become high with respect to the number of data used in learning.
In this paper we examine network performance while reducing the size of the network. The reduction is based on graphical analysis of network output per hidden layer cell and input layer cell. Performance is measured as the sum of squared residuals as well as by the value of largest Lyapunov exponents which is a measure of dynamic instability of time series.

Contents

1 Introduction

Initially neural networks were developed as a simulation model of the brains. The terminology used is still a remainder of this origin. This physiological context, and the fame to handle complex data, may have contributed considerably to the diffusion and implementation of neural network models, also in economics and econometrics; see e.g. Hecht-Nielsen [4], Hertz, Krogh & Palmer [5], Gallant & White [2] and White [14].
In this paper we consider one type of neural network: feed-forward multilayer networks with backpropagation. The analysis is twofold. We examine how the internal structure of the network - the network parameters - works while the neural network attempts to find a relation between in- and output. Secondly, we investigate the performance of the network itself by calculating largest Lyapunov exponents of data sets. Lyapunov exponents measure the sensitivity of the system with respect to small deviations and define as such a characteristic of the given time series, e.g. stability or instability. Moreover, with positive largest Lyapunov exponents, the horizon of prediction is limited since deviations are enhanced. Lyapunov exponent calculations involve the reconstruction of the original data generating process by means of embedded (delayed) data series with the dynamics approximated by a neural network function.
The paper is organized as follows. First we give an introduction to a feed-forward network and how the learning - adaptation of parameters - can be implemented. The second part involves the analysis of the network parameters while learning. We give a graphical procedure to reduce the size of the network. In the third part we apply a neural network to the calculation of Lyapunov exponents.

2 Multi layer feed-forward network

A neural network system consists of neurons (cells), neural interconnections (internal links) and connections with the outer world. In a multi-layer network, neurons are organized in layers with interconnections only between cells of neighboring layers. The network is connected to the outer world by the first or *input* layer and by the last or *output* layer. Between input and output one can have the so-called *hidden* layers. Along the internal links an input signal is fed forward through the hidden layers towards the output layer without feedback.
The number of cells in a layer is called the dimension of the layer; a configuration with an input layer of dimension M, one hidden layer of dimension H, and an output layer of dimension N, is denoted by $nn(M, H, N)$. This type of configuration is applied in this paper. Other configurations are described in Lippmann [8] and Hertz, Krogh and Palmer [5].

2.1 Feed-forward in a multi layer network

The basic concept of a feed-forward neural network, is propagation of a signal from one layer to an other layer. Let $x = (x_1, x_2, \cdots, x_M)'$ be the output signal of an M-dimensional layer; this signal is transmitted to a H-dimensional layer along links with connection weights a_{hm}, $h = 1, 2, \cdots, H$; $m = 1, 2, \cdots, M$ such that the hth cell will receive a signal $a_{hm}x_m$ from the foregoing mth cell. The total signal received by hth cell is equal to $\sum_{m=1}^{M} a_{hm}x_m$. Sensitivity of a cell, also called *internal threshold*, is incorporated in the form of an additional signal b_h; so totally, each cell h receives a signal s_h with

$$s_h = \sum_{m=1}^{M} a_{hm}x_m + b_h, \ h = 1, 2, \cdots, H.$$

Note that thresholds are automatically incorporated if the input signal has the form $x = (1, x_1, x_2, \cdots, x_M)'$.
How signal s_h is propagated depends on the *activation function* of cell h. This activation function, denoted as g, will be monotone and bounded; commonly used is the logistic function

$$g(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

So, the output signal of cell h equals to

$$g(\sum_{m=1}^{M} a_{hm}x_m + b_h). \tag{2}$$

Equation (2) describes one pass through one layer cell. In matrix form, a layer with H-cells transmits the output signal $x = (x_1, \cdots, x_M)$ of the foregoing layer, as:

$$G(Ax + B) \tag{3}$$

where

$$\begin{aligned} A &= [a_{hm}], \ H\text{x}M \text{ matrix of connection weights,} \\ B &= (b_1, \cdots, b_H)', \text{ vector of internal thresholds,} \\ G(x_1, \cdots, x_H) &= (g(x_1), \cdots, g(x_H))', \ G : \mathbf{R}^H \to \mathbf{R}^H, \end{aligned}$$

where g is an activation function; see e.g. equation (1). A multi-layer feed-forward network consists of consecutive layers with an input-output relation given by equation (3). For a $nn(M, H, N)$ three layer network with no transformations at input-layer and output-layer level, the relation between an input signal $x = (x_1, \cdots, x_M)'$ and an output signal $\hat{y} = (\hat{y}_1, \cdots, \hat{y}_N)'$ is given by

$$\hat{y} = C\, G(Ax + B) \tag{4}$$

where C is the matrix of $N\text{x}H$ connections weights between hidden layer and output layer.

Depending on the activation function and sensibility of cells, different configurations can be implemented. For instance, a mixture of linear and non linear interaction is given if part of the hidden layer cells have linear activation functions which results in an output signal equal to

$$\hat{y} = Dx + E + CG(Ax + B).$$

2.2 Supervised learning, backpropagation

A neural network is designed to fulfill some task, e.g. to find an approximation of a supposed relation $y = F(x)$, $x \in \mathbf{R}^M$, $y \in \mathbf{R}^N$. In this section we describe the so-called *learning* process.
Given the structure of the network and an input x, a network will produce an output $\hat{y}$. The output depends on all connection weights and thresholds which we will call the parameters of the network. In the case of so-called *supervised learning* network output $\hat{y}$ is compared with actual (or desired) output y for all input vectors x involved; here learning means a sequential process of parameter adjustments such that the error, difference between $\hat{y}$ and y, becomes smaller with respect to some norm. The implementation of the learning process is one of the characteristics of the network.
In a multi layer neural network with supervised learning, one defines learning as decreasing the sum of squared errors (residuals) at each layer using parameter adjustment by gradient descent; see e.g. Amari [1]. With H hidden layers, we discriminate between quantities and qualities of layers by a subscript h, where $h = 0$ is the input layer and $h = H + 1$ is output layer. We use the following notations:

$$\begin{aligned}
x &:= \text{network input vector,} \\
A_h &:= \text{matrix of connection weights between layer } h-1 \text{ and layer } h, \\
B_h &:= \text{vector of internal thresholds of layer } h \text{ cells,} \\
\hat{y}_h &:= \text{network output vector at layer } h, \text{ so } \hat{y}_h = G(A_h \hat{y}_{h-1} + B_h), \\
y_h &:= \text{vector of desired outputs at layer } h. \\
e_h &:= \text{error vector at layer level } h, \text{ so } e_h = \hat{y}_h - y_h.
\end{aligned}$$

First we describe the parameter adjustment in one layer, with input-output relation

$$\hat{y}_h = G(A_h \hat{y}_{h-1} + B_h). \tag{5}$$

In this case desired output is equal to y_h so an error vector e_h is computed as $\hat{y}_h - y_h$. At this level, the object function is defined as the sum of squared errors, SSR_h, with

$$SSR_h = (G(A_h \hat{y}_{h-1} + B_h) - y_h)'(G(A_h \hat{y}_{h-1} + B_h) - y_h).$$

The state of the learning process at time t of the neural net is embodied in the value of the parameters A_h and B_h at time t. We refer to this knowledge at time t, if necessary,

by an additional subscript t.
Now learning is given by a sequential process in time:

$$A_{h,t+1} = A_{h,t} + \Delta A_{h,t} \tag{6}$$
$$B_{h,t+1} = B_{h,t} + \Delta B_{h,t} \tag{7}$$

where the adjustments ΔA_h and ΔB_h are chosen such that SSR_h will decrease. In general, gradient-descent steps are used:

$$\Delta A_h = -\eta \frac{\partial SSR_h}{\partial A_h} \tag{8}$$
$$= -\eta (D_x G_{(A_h \hat{y}_{h-1} + B_h)})' (G(A_h \hat{y}_{h-1} + B_h) - y_h)(\hat{y}_{h-1})' \tag{9}$$

$$\Delta B_h = -\eta \frac{\partial SSR_h}{\partial B_h} \tag{10}$$
$$= -\eta (D_x G_{(A_h \hat{y}_{h-1} + B_h)})' (G(A_h \hat{y}_{h-1} + B_h) - y_h) \tag{11}$$

where $D_x G_{(A_h \hat{y}_{h-1} + B_h)}$ is the gradient of G evaluated at $A_h \hat{y}_{h-1} + B_h$. The parameter η is called the *learning rate* and has to be chosen properly [1].
The whole adjustment procedure is based on the existence of a desired output y_h at each layer h of the network. In general with one or more hidden layer units, only at output level an error vector can be found directly by comparing network output with desired output. To solve this problem, one applies *backpropagation* of the error vector which can be derived as follows.
Suppose the error vector $e_h = G(A_h \hat{y}_{h-1} + B_h) - y_h$ at layer h exists. To find the error e_{h-1} in layer $h-1$ output, we have to define a desired output y_{h-1} at this layer. The desired value of y_{h-1} is taken to be such that the object function SSR_h (at layer h) has a smaller value with input y_{h-1} than with input $\hat{y}_{h-1}$. So with $y_{h-1} = \hat{y}_{h-1} - e_{h-1}$, y_{h-1} is chosen such that:

$$(G(A_h(\hat{y}_{h-1} - e_{h-1}) + B_h) - y_h)'(G(A_h(\hat{y}_{h-1} - e_{h-1}) + B_h) - y_h) \tag{12}$$

is less or equal to

$$(G(A_h \hat{y}_{h-1} + B_h) - y_h)'(G(A_h \hat{y}_{h-1} + B_h) - y_h). \tag{13}$$

After expansion of equation (12) in the (small) error e_{h-1}, one gets the following condition on e_{h-1}:

$$-2(D_x G_{(A_h \hat{y}_{h-1} + B_h)} A_h e_{h-1})'(G(A_h \hat{y}_{h-1} + B_h - y_h) + \text{ higher order terms in } e_{h-1} \leq 0. \tag{14}$$

[1] A refinement of the updating equation is the use of a time delay. Denoting the adjustments calculated in equation (9) and (11) by ΔA_{h*} and ΔB_{h*} and using the time subscript t one defines: $\Delta A_{h,t} = \Delta A_{h*,t} + m \Delta A_{h,t-1}$ where m is the so-called *momentum* parameter; and similar for $\Delta B_{h,t}$. The use of time-delay avoids erroneous wandering of the parameters which may occur if the learning parameter η is too large.

A solution is to take e_{h-1} equal to

$$A_h'(D_x G_{(A_h \hat{y}_{h-1}+B_h)})'(G(A_h \hat{y}_{h-1} + B_h - y_h). \tag{15}$$

Hence e_{h-1} can be found according to the following equation:

$$e_{h-1} = A_h'(D_x G_{(A_h \hat{y}_{h-1}+B_h)})'\, e_h. \tag{16}$$

Equation (16) defines the backwards propagation of an error vector through the network. Since at output level the error vector is well defined in a supervised learning network, the *backpropagation* mechanism provides an error vector at each layer of the network. The adjustments of the parameters A_h and B_h are now calculated according to equation (9) and (11).
Finally, if the error vector has reached the input layer, all adjustments can be calculated and the parameters of the network are updated. So one step in the learning process consists of one forward pass of an input vector and one backward pass of the output error vector with calculation of the parameter adjustments by equations (9) and (11).
We note that this updating procedure is just a gradient descent method for minimizing the network squared error function $SSR(x) = (\hat{y} - y)'(\hat{y} - y)$ for one input vector x. Suppose the network is given as $CG(Ax + B)$: one hidden layer and no transformations at input- and output level. Using the foregoing notations, desired output y_2 is equal to y, network output is equal to $\hat{y}_2$ and e_2 is the error vector at output level. Moreover $A_2 = C$ and $B_2 = 0$. Hence

$$e_2 = \hat{y}_2 - y = A_2 \hat{y}_1 + B_2 - y = C\hat{y}_1 - y,$$

where

$$\hat{y}_1 = G(Ax + B).$$

The adjustment ΔC is given by equation (8) with $h = 2$, $A_2 = C$, $B_2 = 0$ and $\Delta A_2 = \Delta C$. Consequently, applying ΔC will decrease the network squared error function $SSR(x)$ where

$$SSR(x) = (C\hat{y}_1 - y)'(C\hat{y}_1 - y) = (CG(Ax + B) - y)'(CG(Ax + B) - y).$$

At hidden layer level 1, the error vector e_1 is given as (see equation (16))

$$e_1 = C'\, e_2 = C'\,(C\hat{y}_1 - y) = C'\,(CG(Ax + B) - y). \tag{17}$$

Hence y_1, desired output at level 1, equals $\hat{y}_1 - e_1 = G(Ax + B) - e_1$. So according to equations (9) and (11), the adjustment ΔA is given as

$$\begin{aligned} \Delta A &= -\eta \frac{\partial e_1' e_1}{\partial A} && (18)\\ &= -\eta \frac{\partial (G(Ax + B) - y_1))'(G(Ax + B) - y_1)}{\partial A} && (19)\\ &= -\eta D_x G_{(Ax+B)}\, e_1\, x' && (20) \end{aligned}$$

and similar for ΔB:

$$\Delta B \;=\; -\eta D_x G_{(Ax+B)}\, e_1 \tag{21}$$

Substituting $e_1 = C'\, e_2 = C'(CG(Ax+B) - y)$ in those equations, one finds

$$\begin{aligned}
\Delta A &= -\eta D_x G_{(Ax+B)} C'(CG(Ax+B) - y)x' && (22)\\
\Delta B &= -\eta D_x G_{(Ax+B)} C'(CG(Ax+B) - y) && (23)\\
\Delta C &= -\eta (CG(Ax+B) - y)(G(Ax+B))' && (24)
\end{aligned}$$

Hence the adjustments obtained by the backpropagation scheme, are equal to one gradient descent step applied to the object function $SSR(x) = (CG(Ax+B)-y)'(CG(Ax+B)-y)$.

2.3 Nonlinear least squares

The iteration step exposed above, involves only one input vector and one related output vector. Suppose one has a set X of input vectors x_t, $t = 1, \cdots, T$ and a set Y of output vectors y_t. In order that a neural network learns the assumed relation $y_t = F(x_t)$, input vectors x are chosen at random from the given set X and for each input, the network parameters are adjusted according to the foregoing procedure. [2] In this way, the learning procedure is a iterative nonlinear least squares method which will minimize the total error function, the sum of squares of the residuals SSR

$$SSR = \sum_{t=1}^{T} (CG(Ax_t + B) - y_t)'(CG(Ax_t + B) - y_t). \tag{25}$$

As we are not interested in the neural network as a model of learning, we apply a nonlinear optimization procedure to find the parameters A, B and C which minimize SSR given by equation (25). To be specific: we apply a variable metric method known as Davidon-Fletcher-Powell; see Press e.a., [9]. However to avoid that the system gets stuck initially in some small local minimum, we apply first the backpropagation learning procedure on randomly chosen items of the learning set with learning parameter η not too small. As soon as this procedure does not show any progress anymore, even with small learning parameter η, we continue with the variable metric method.

3 Reconstruction data generating process

Suppose one observes a one dimensional time series x_t, $t = 1, \cdots, T$. In general, the series x_t is generated by a more dimensional system. Suppose the time series x_t has a deterministic explanation: the data generating process can be written as $z_t = G(z_{t-1})$

[2] Generally the input and output set is divided in two subsets: one is the so-called learning set while the remaining input and output vectors are used for testing the knowledge of the learned network.

and $x_t = h(z_t)$ where $G : \mathbf{R}^N \to \mathbf{R}^N$ and $h : \mathbf{R}^N \to \mathbf{R}$; see Takens [12]. Now the unknown N dimensional dynamical system can be reconstructed generally by the time evolution of the M dimensional embedded vectors $x_t^M = (x_t, x_{t+1}, \cdots, x_{t+M-1})$ where M is called the embedding dimension. According to Takens [12], the dynamical system

$$x_t^M = \Psi(x_{t-1}^M), \tag{26}$$

$$\Psi = (\psi^1, \cdots, \psi^M), \text{ with } \psi^m : \mathbf{R}^M \to \mathbf{R} \tag{27}$$

is a reconstruction of the unknown system $z_t = G(z_{t-1})$ if $M \geq 2N + 1$.
Note that the only unknown component of Ψ is the Mth component function ψ^M since

$$\psi^m(x_1, \cdots, x_M) = x_{m+1}, \ m = 1, \cdots, M-1.$$

Hence the reconstruction involves only the estimation of the Mth component function ψ^M.
For a given time series x_t one has the relation

$$x_{t+M} = \psi^M(x_t, \cdots, x_{t+M-1}). \tag{28}$$

The function ψ^M is now approximated by $\hat{\psi^M}$ as a neural network with M input neurons, H hidden layer neurons and 1 output neuron, a $nn(M, H, 1)$ network.

$$\hat{\psi^M}(x^M) = CG(Ax^M + B). \tag{29}$$

Equation (28) defines an auto-regressive process; in linear approximation, an $AR(M)$ process.
The reconstruction procedure involves still one major problem: the unknown dimension N of the original system or equivalently, the unknown embedding dimension M. Since for M large, $M \geq 2N + 1$, the characteristics of the embedded system will not alter, one way out is to calculate a quantity which is a characteristic of the original dynamical system and which is an invariant of embedding. By doing this for increasing embedding dimensions M, the lowest value of M where this quantity will not alter, is the correct embedding dimension. A possible characteristic value is the largest Lyapunov exponent which we will introduce now.

3.1 Lyapunov exponents

Suppose $z_t = G(z_{t-1}), G : \mathbf{R}^N \to \mathbf{R}^N$, then under general conditions; see [3], the largest Lyapunov exponent λ can be calculated as

$$\lambda = \lim_{T \to \infty} \lambda(T), \tag{30}$$

$$\lambda(T) = \frac{1}{T} \ln \| D_z G^T(z)\, w \|, \text{ for almost all } w \in \mathbf{R}^N, \tag{31}$$

where $D_z G^T$ is the gradient of the Tth iteration of G; see [3]. If the dynamic function G is given, one computes $\lambda(T)$ as follows:

$$\lambda(T) = \frac{1}{T} \ln\{\prod_{t=1}^{T} \frac{\|D_z G(z_{t-1}).w_{t-1}\|}{\|w_{t-1}\|}\}, \tag{32}$$

$$= \frac{1}{T} \sum_{t=1}^{T} \ln\{\frac{\|D_z G(z_{t-1}).w_{t-1}\|}{\|w_{t-1}\|}\}, \tag{33}$$

$$z_t = G(z_{t-1}), \tag{34}$$

$$w_t = D_z G(z_{t-1})\, w_{t-1},\ t = 1, \cdots, T,\ w_0 \text{ given.} \tag{35}$$

The initial vector w_0 is chosen arbitrary[3]. With only finite data available, one uses $\lambda(T)$ as an approximation of λ.
Although equation (33) involves only time steps of one unit, the largest Lyapunov exponent can be calculated also with larger time steps. For a time step equal to k units, one gets

$$\lambda(T) = k\, \lambda_k(T), \tag{36}$$

$$\lambda_k(T) = \frac{1}{T} \ln\{\frac{\|D_z G^k(z_{t-k}).w_{t-k}\|}{\|w_{t-k}\|}. \tag{37}$$

As a consequence, the quantities $\lambda_k(T)$ are linearly related in k. The graph of $\lambda_k(T)$ against time-steps k has to be linear (in the range of time-steps where expansion is not limited by the extension of the attractor). The largest Lyapunov exponent λ is estimated from this graph as the slope of the linear (=straight) part of the graph. This procedure avoids the problem of spurious large exponents at too low embedding dimensions; see [6].

The largest Lyapunov exponent λ measures the logarithmic rate of expansion of an initial deviation w_0 along the solution z_t of $z_t = G(z_{t-1})$. If λ is positive one has an unstable orbit (time series): deviations will expand according to $e^{\lambda\, t}$ and consequently predictions have limited time-horizon.

4 Analysis of network learning

In this section we discuss the performance of a neural network. We will examine for each hidden layer cell and each input layer cell the contribution to total network output. The analysis is based on a $nn(4, 6, 1)$ network with input a 4-dimensional embedded vector given as $(x_t, x_{t-1}, x_{t-2}, x_{t-3})$ while desired output is given by x_{t+1}. The data set x_t is the Nelson-Plosser time series on unemployment in the USA with 99 observations[4]. This data set is denoted by $UNEMPLOY$. All data are scaled down to a $[0, 1]$ range.

[3]Since equation (33) with $T \to \infty$ will give a correct value of the largest Lyapunov exponent for almost all w_0, it makes sense to repeat the calculations for different w_0.

[4]See Schotman & van Dijk [10] for an extensive description of the data.

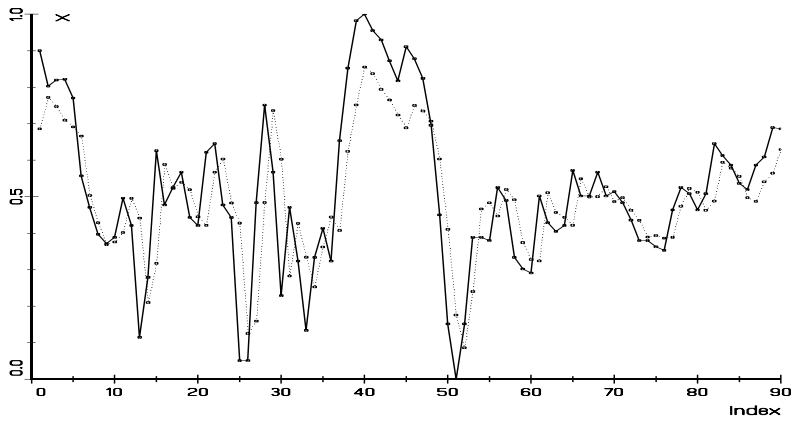

Figure 1: Actual series (continuous curve) and prediction (one period ahead) from a $nn(4,6,1)$ network; early stage of learning

The optimization (= learning) procedure is started with randomly chosen parameters values. Already after a few (25) iterations the network output compares well with the desired output although the dotted (= estimated data) curve seems to be one period behind the continuous (= actual data) curve; see figure 1.
In figure 2 we show neural network output with the contribution of one input cell left out: a graph with label $I_n, n = 1,2,3,4$ displays neural network output minus the contribution of input cell n (all network connections to input cell n are broken). This procedure is comparable with the analysis of partial contribution of linear terms in Theil, pag. 185 [13]. It is clear that the first input cell, the x_t data, can not be neglected, while the contribution of all other input cells is hardly significant.

In figure 3 we show network output of each hidden layer cell separately (connections from output with all other hidden layer cells are broken). At this stage, only hidden layer cell 1 produces an output pattern comparable with actual data; output of all other cells are more or less constant on the whole range of inputs. Especially the output of cell 3 is zero and does not contribute to total output at all.

Figures 4, 5 and 6 show the performance of the network at the final stage of optimization. The picture has changed considerable. The predicted values are no longer lagged by one period; see figure 4. The contribution of all input cells is significant; see figure 5. However, the output of the majority of hidden layer cells resembles a block function: the cells react only to specific input values with a fixed response; see figure 6.
A second aspect is that hidden layer cells 3 and 5 have a similar output pattern, only

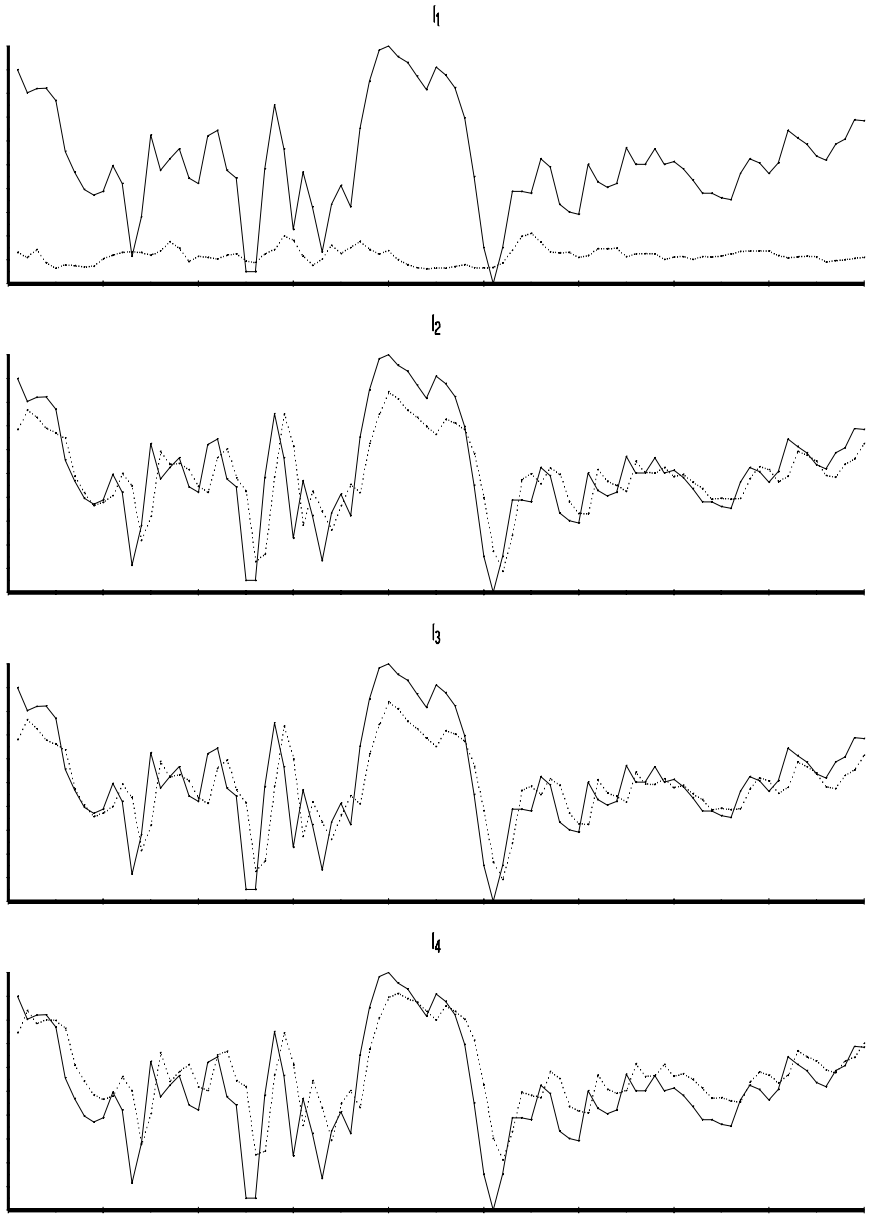

Figure 2: Network output with the contribution of one single input cell left out (dotted curve) compared to actual data (continuous curve); early stage of learning

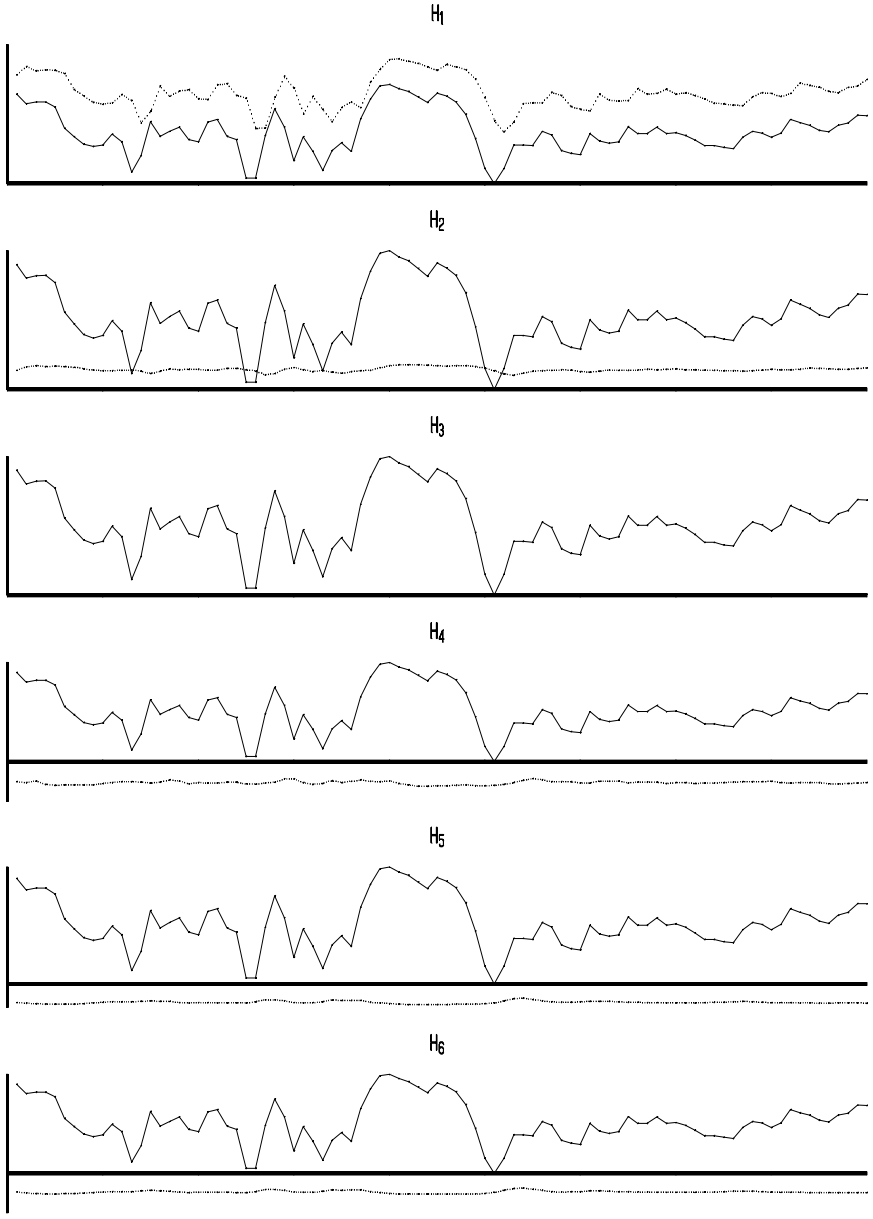

Figure 3: Output of one hidden layer cell (dotted curve) compared to actual data (continuous curve); early stage of learning

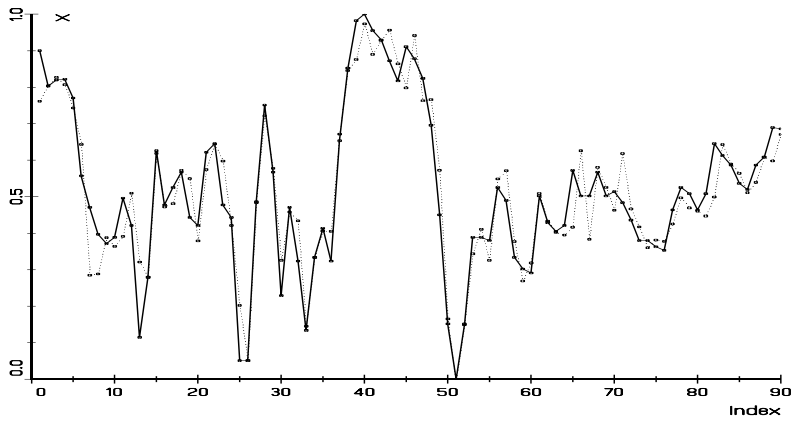

Figure 4: Actual series (continuous curve) and estimates series (dotted curve) from a $nn(4,6,1)$ network; final stage of learning

reversed in sign. So it seems possible to reduce the size of the network by deleting one of these hidden layer cells. The results of this reduced $nn(4,5,1)$ network is displayed in figure 7. Now all hidden layer cells have different output patterns[5].

If more hidden layer cells are added to the network, the output patterns will look more and more like single bump functions, similar to the output of hidden layer cell 4 in figure 7. Moreover, redundancy becomes apparent in two ways:

- Two layers may have similar output patterns but just reversed in sign; the contribution to total output of those layers is very small; see e.g. figure 8. This allows also for a reduction in the number of parameters by sequential reducing the number of hidden layers cells till the network output becomes significantly affected.

- The response pattern of individual hidden layer cells resembles (a sum of) δ-like functions; it is clear that prediction (generalisation) by such a neural network will be weak. Already by the nature of neural network, especially the range of the sigmoid functions, prediction on values outside the learning data range is limited. This effect will be enhanced if the neural network degenerates to a sum of δ-like functions.

[5]The SSR of the $nn(4,6,1)$ network is 0.425 while the reduced $nn(4,5,1)$ network has $SSR = 0.539$.

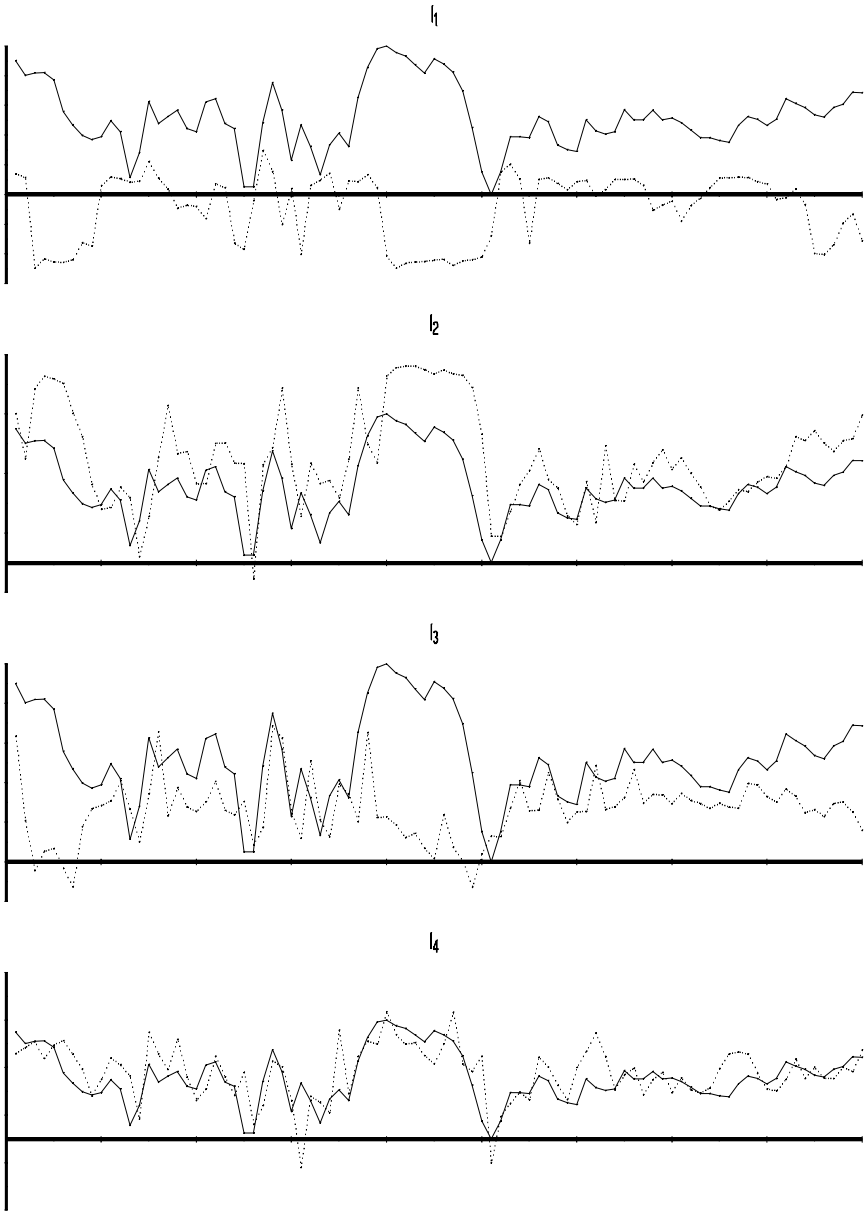

Figure 5: Network output with the contribution of one single input cell left out (dotted curve) compared to actual data (continuous curve); final stage of learning

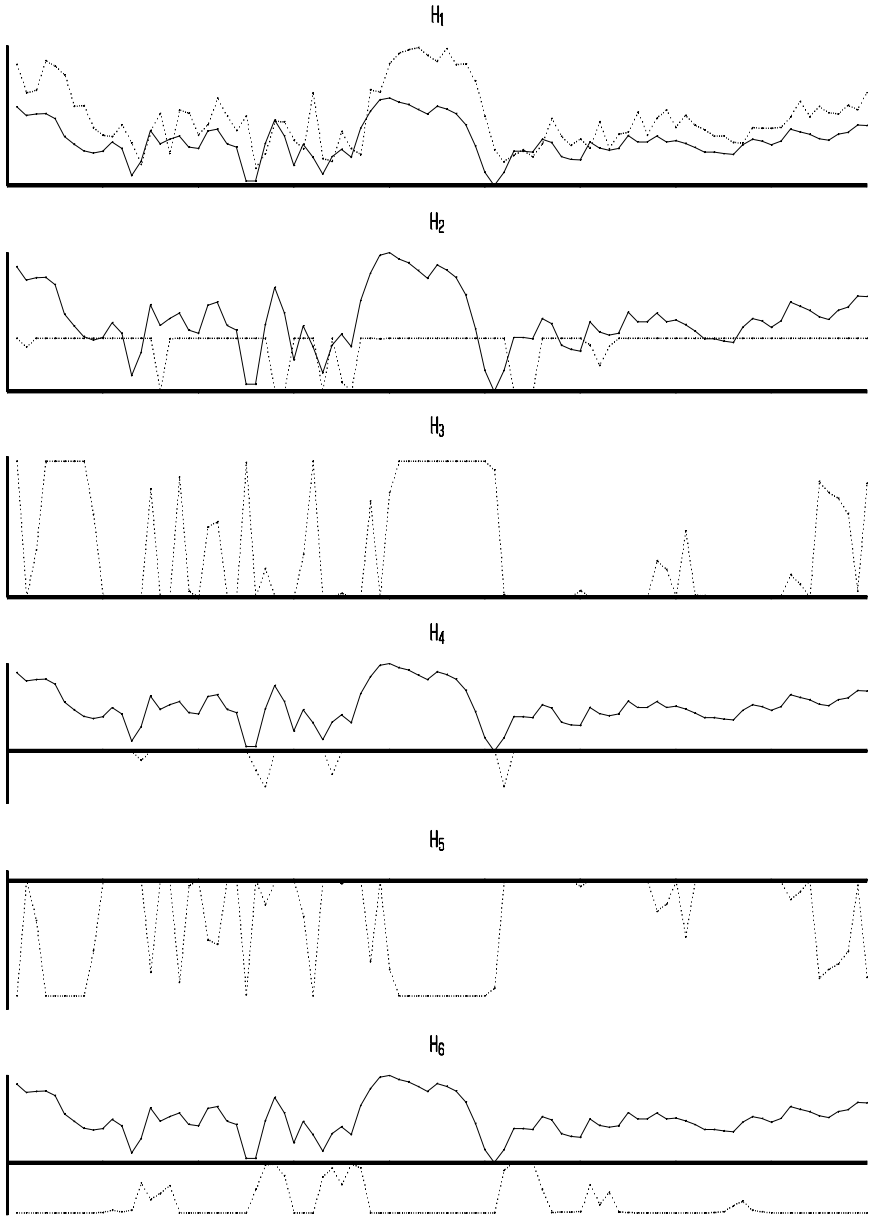

Figure 6: Output of one hidden layer cell (dotted curve) compared to actual data (continuous curve); final stage of learning

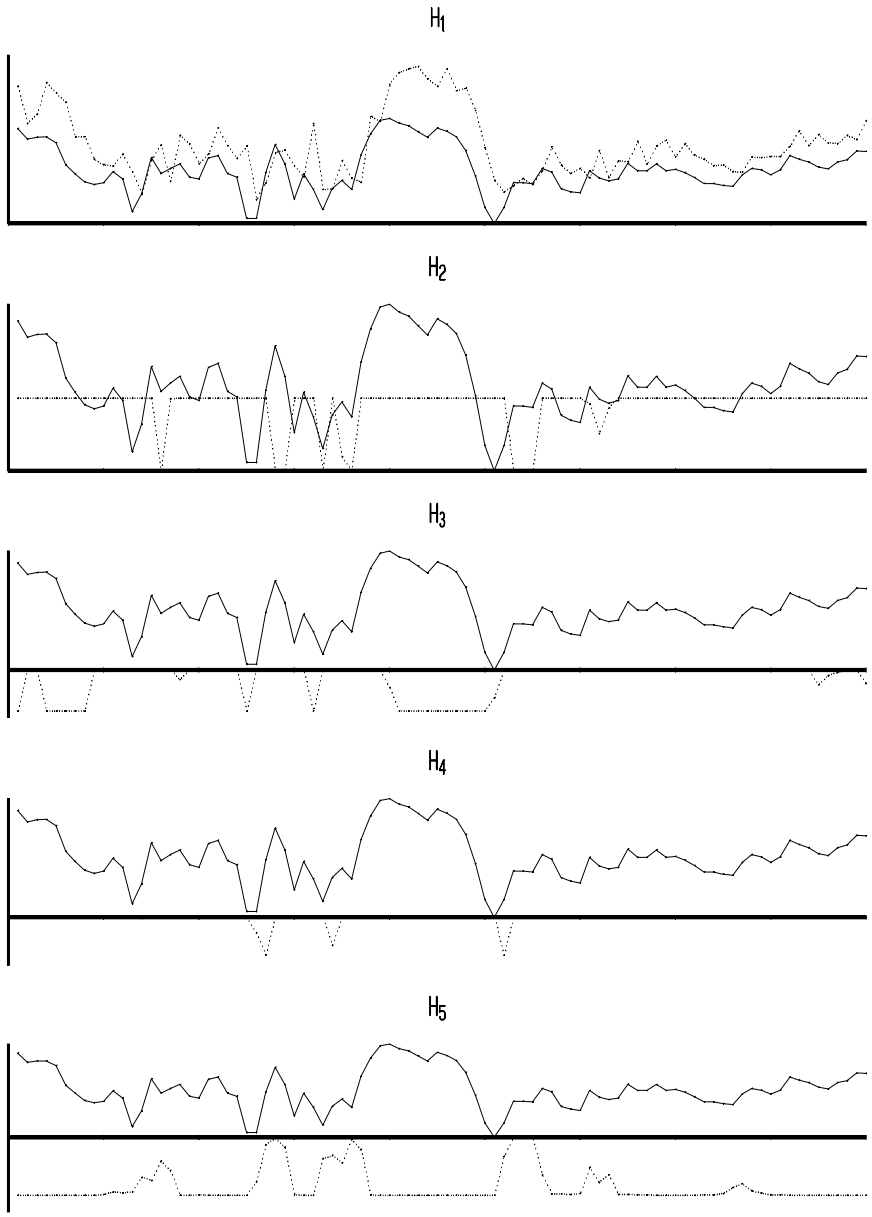

Figure 7: Output of one hidden layer cell (dotted curve) compared to actual data (continuous curve) in a $nn(4,5,1)$ network; final stage of learning

5 Lyapunov exponent

In this section we give some results on the calculation of the largest Lyapunov exponent for different data sets. The Lyapunov exponent is calculated along a given orbit (time series); e.g. in equation (32) the term z_{t-1} represents an element of the orbit. So one can do the calculations in two ways: either use a given time series or use a time series generated by a neural network as an approximation of the data generating function. In the latter case, the initial start vector is taken from the given data set but the next data are generated according to equation (26) and equations (28) and (29). The largest Lyapunov exponent calculated this way will be denoted as $\lambda(nn)$ while the exponent calculated from the given time series is denoted as $\lambda(ts)$. If the neural network has learned correctly the structure of the data generating process then both largest Lyapunov exponents would be the same. Here we refer to structure as the deterministic component of the data generating process. Although already in a deterministic, chaotic process with positive largest Lyapunov exponent, nearby starting orbits will differ in time and are uncorrelated (limited forecasting horizon), similarity between Lyapunov exponents based on given time series and based on neural network series, will mean that one of the characteristics of the data generating process is learned by the neural network. Hence the difference between $\lambda(ts)$ and $\lambda(nn)$ is a measure of learning.
We will also report the value of the object function SSR, sum of squared residuals and R^2, the squared correlation coefficient between actual data and network output. The performance on test data is measured as the mean sum of squared residuals, $MSSR(test)$. Abundance of parameters is measured by the so-called information criterion, SIC, defined as

$$SIC = \ln(MSSR) + \frac{n_p}{2T} ln(T), \tag{38}$$

where n_p is the number of parameters, T is the length of data set and $MSSR$ is the mean sum of squared residuals; see Schwartz [11].

In all cases we start with a high number of hidden layer cells, e.a. $H = 10$. As proposed in the foregoing section, this number is reduced by deleting hidden layer cells which respond to only one or few inputs and/or have similar output patterns. However it is known that a correct estimation of Lyapunov exponents require a high degree of correlation between actual and estimated series; see [6].

The first data are the $UNEMPLOY$ series from the foregoing section with the final 5 data used as test data. The results with $H = 10$ are summarized in table 1.
The Lyapunov exponents don't show a convergence to a constant value with increasing embedding dimension. This may well be caused by the possibility that with a high number of hidden layer cells, the network learns not only about a deterministic component but also about a stochastic component in the data. In that case, one can not find an embedding dimension where the Lyapunov exponent becomes stabilized; a stochastic process does not allow for a finite embedding. Because of the steep decrease in SSR and the similarity between Lyapunov exponent of the time series and of the orbit, we consider

Table 1: Neural network results on data set $UNEMPLOY$

network		results learning data					results test data
M	H	SSR	R^2	SIC	$\lambda(ts)$	$\lambda(nn)$	$MSSR$
1	10	1.3700	0.6817	-1.3778	0.11	-1.05	0.0116
2	10	0.8237	0.8059	-1.3468	1.05	-0.05	0.0031
3	10	0.3529	0.9171	-1.5369	1.23	-0.67	0.0064
4	10	0.3417	0.9175	-1.2492	1.24	–.–	0.0065
5	10	0.0571	0.9856	-1.8671	1.38	1.17	0.0424
6	10	0.0337	0.9915	-1.8977	1.07	0.44	0.0116
7	10	0.0139	0.9963	-2.0006	0.79	0.46	0.0329

embedding dimension 5 as sufficient[6].
In figure 8 is shown the output of each hidden layer cell of the $nn(5, 9, 1)$ network together with the actual data. It is obvious that several hidden layers have similar output, only different in sign and/or scale. Now we reduce the net, either by letting out the hidden layer with smallest contribution to total output, or two hidden layers at the same time if the output of the layers are similar and only reverse in sign. The reduction process is summarized in table 2.

Table 2: Neural network results on data set $UNEMPLOY$

network		results learning data					results test data
M	H	SSR	R^2	SIC	$\lambda(ts)$	$\lambda(nn)$	$MSSR$
5	9	0.0812	0.9796	-1.8713	1.17	0.91	0.0121
5	8	0.0952	0.9761	-1.9713	1.27	0.91	0.0324
5	6	0.3852	0.9035	-1.6320	0.60	-0.15	0.0020
5	4	0.6011	0.8494	-1.7688	0.34	0.41	0.0263

Note that with 6 hidden layer cells, the orbit has a negative Lyapunov exponent. This orbit is periodic. In all other cases, the Lyapunov exponents of the time series and the orbit are similar. As noted before, we observe with increasing SSR a decrease in the Lyapunov exponents. However in all but one cases, similarity between time series and orbit, generated by the neural network, is preserved. The largest Lyapunov exponents along the given time series and the orbit are of the same order.

[6]For the $nn(4, 10, 1)$ network, the orbit diverges, so the largest Lyapunov exponent could not be calculated.

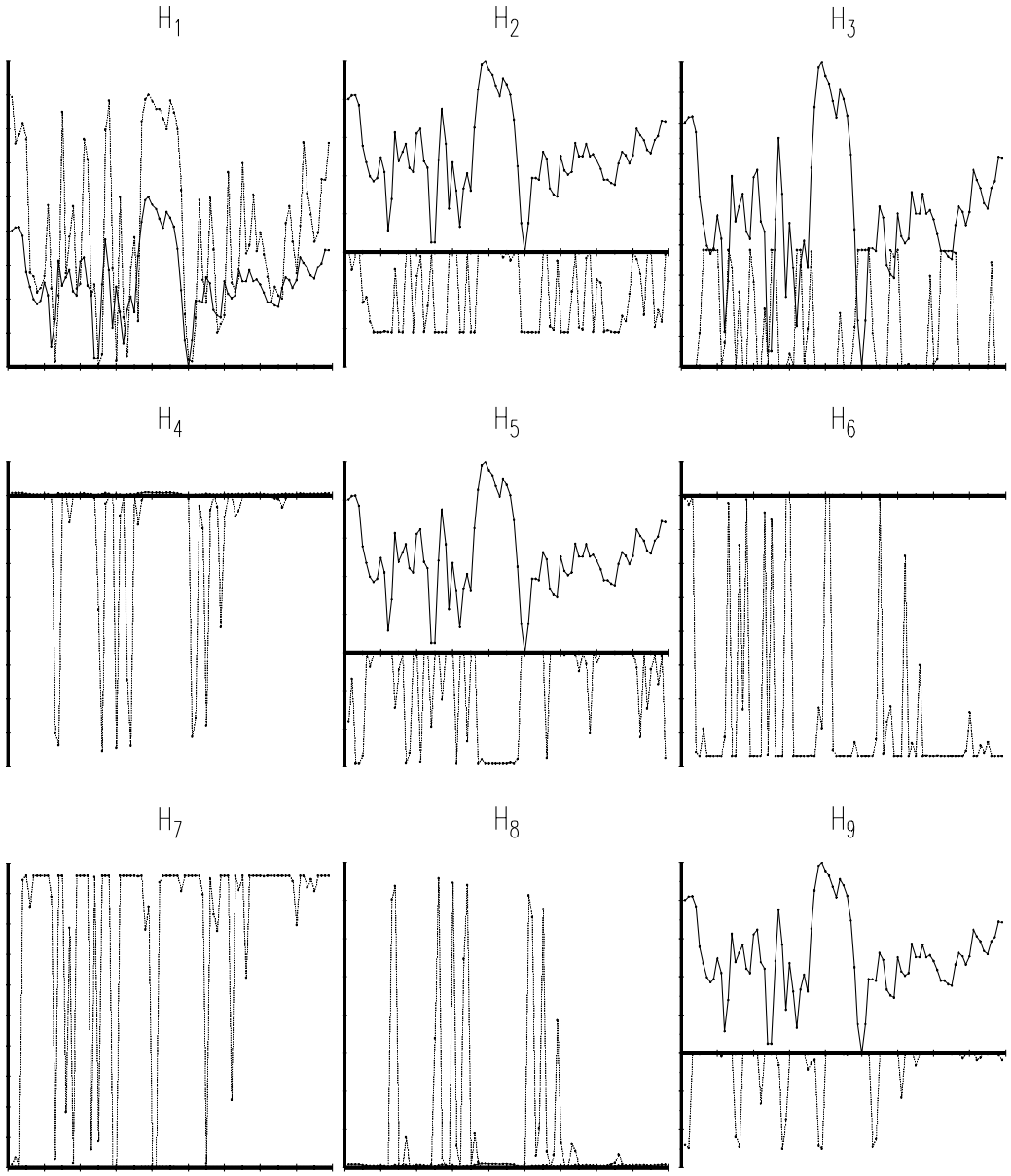

Figure 8: Hidden layer output of a $nn(5, 9, 1)$ network applied to data $UNEMPLOY$; final stage of learning

The second data set contains the long term and short term interest rate in USA; see [7]. The data sets are denoted by LT, long term interest rate, and ST, short term interest rate. The sample period is from January 1957 till April 1989. The length of data series (388) allows for a longer forecasting period. We restrict network learning to the first 300 data and use the rest as test data. The results for the long term interest rates are summarized in table 3 while results on short term interest rates are reported in table 4.

Table 3: Neural network results on data set LT

network		results learning data					results test data
M	H	SSR	R^2	SIC	$\lambda(ts)$	$\lambda(nn)$	$MSSR$
2	5	0.0956	0.9930	-3.8350	-0.04	-0.06	0.0018
3	5	0.0741	0.9947	-3.9151	0.08	-0.07	0.0038
4	5	0.1086	0.9923	-3.7240	-0.01	-0.50	0.0014

Table 4: Neural network results on data set ST

network		results learning data					results test data
M	H	SSR	R^2	SIC	$\lambda(ts)$	$\lambda(nn)$	$MSSR$
2	5	0.3268	0.9699	-3.2209	0.01	-0.10	0.0040
3	5	0.3189	0.9712	-3.1855	0.10	-0.10	0.0026
4	5	0.3075	0.9726	-3.1564	0.18	-0.51	0.0027

We stopped at embedding dimension 4; we didn't find an improvement in the SSR at higher embedding dimension. In all cases the Lyapunov exponents are near to zero. Without statistical analysis of those estimates, it is difficult to make any decision about the real sign. A linearization of the neural network function, results in a AR process with coefficients sum up to 1 (approximately); this agrees with the results of Kleibergen & van Dijk [7].
Since both time series are observations from one and the same dynamic process, one can argue that both series are generated by the same dynamic function. So we constructed a neural network with input vectors build from delayed long- and short term rent data while actual output is given by a two-dimensional vector of long- and short term rents of the next period. So the structure of the network is $nn(2M, H, 2)$ where M is the number of lags (embedding dimension) applied to each data series separately. The results are summarized in table 5.

Table 5: Neural network results on bivariate input LT and ST

network		results learning data						results test data
M	H	SSR	$R^2(LT)$	$R^2(ST)$	SIC	$\lambda(ts)$	$\lambda(nn)$	$MSSR$
1	5	0.6052	0.99	0.96	-2.8654	0.01	-0.13	0.0197
2	5	0.3618	0.99	0.98	-3.0274	0.08	-0.05	0.0131
3	5	0.3044	0.99	0.98	-3.0189	0.08	-0.05	0.0105

In this case an embedding dimension of 2, two time lags for each variable, seems to be proper. The largest Lyapunov exponent of the bivariate system is positive which is to be expected since at least one of the series itself has a positive largest Lyapunov exponent.

6 Conclusions

At least two problems arise applying a neural network in the reconstruction of a data generating process. First, the original system can be highly dimensional or a mixture of a low dimensional process with a high dimensional stochastic component. This means that the proper embedding dimension is either very large or hardly to find.
Second, one does not know the size of the hidden layer. As one would like to have a parsimonious system, one chooses the number H of hidden layer cells as small as possible. However a characteristic quantity like the largest Lyapunov exponent requires a high degree of similarity between actual data and the neural network as data generating function. So one is tempted to use a network with a high number of hidden layer cells. However, as we have shown in this paper, redundancy in the number of hidden layer cells is reflected in δ-like output patterns of some of the hidden layer cells.
In this paper we show that graphical analysis of hidden layer cell contribution to total output may provide a way to reduce the size of the network. Applied to economic data, reduction is possible while preserving similarity between given time series and orbits generated by the estimated neural network function.

References

[1] Amari, S., Mathematical Foundations of Neurocomputing, Proceedings of the IEEE, vol.**78**,no.9,, 1990.

[2] Gallant, A.R. & H. White, There exists a neural network that does not make avoidable mistakes, in Proc. of the International Conference on Neural Networks, San Diego, 1988, IEEE Press, New York, 1989.

[3] Guckenheimer, J. & P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer-Verlag, New York - Berlin - Heidelberg - Tokyo, 1983.

[4] Hecht-Nielsen, R., *Neurocomputing.* Addison-Wesley Publ. Co., Menlo Park, CA, 1990.

[5] Hertz, J., A. Krogh & R.G. Palmer, *Introduction to the theory of neural computation,* Addison-Wesley Publishing Company, Reading Massachusetts, 1991.

[6] Kaashoek, J.F. & H.K. van Dijk, Evaluation and Application of numerical procedures to calculate Lyapunov exponents, Econometric Reviews, special issue, Vol.13, No.1, 1994..

[7] Kleibergen, F.R. & H.K. van Dijk, Bayesian analysis of a *GARCH* model, to appear in Journal of Applied Econometrics, 1993.

[8] Lippmann, R.P., An Introduction to Computing with Neural Nets, IEEE ASSP Magazine, April 1987.

[9] Press, W.H., B.P. Flannery, S.A. Teukolsky & W.T. Vetterling, *Numerical Recipes.* Cambridge University Press, Cambridge, 1988.

[10] Schotman, P. & H.K. van Dijk, On Bayesian routes to unit roots, Journal of Applied Econometrics, 1991

[11] Schwartz, G., Estimating the Dimension of a Model, The Annals of Statistics, **6**, 1978.

[12] Takens, F., Detecting strange attractors in turbulence, in D.A. Rand and L.S. Young (eds.), Dynamical systems and turbulence. Springer-Verlag, Berlin, 1981.

[13] Theil, H., *Principle of Econometrics*, Wiley & Sons, 1971

[14] White, H., Some Asymptotic Results for Learning on Single Hidden Layer Feedforward Network Models, Journal of the American Statistical Association,vol.**84**, no.408, 1989.

[15] Wolf, A., J.B. Swift, H.L. Swinney & J.A. Vastano, Determining Lyapunov exponents from a time series. Physica 16D. 285-317, 1985.