International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:2, No:6, 2008

System Identification with General Dynamic Neural Networks and Network Pruning

Christian Endisch, Christoph Hackl, and Dierk Schröder

***Abstract*—**This paper presents an exact pruning algorithm with adaptive pruning interval for general dynamic neural networks (GDNN). GDNNs are artificial neural networks with internal dynamics. All layers have feedback connections with time delays to the same and to all other layers. The structure of the plant is unknown, so the identification process is started with a larger network architecture than necessary. During parameter optimization with the Levenberg-Marquardt (LM) algorithm irrelevant weights of the dynamic neural network are deleted in order to find a model for the plant as simple as possible. The weights to be pruned are found by direct evaluation of the training data within a sliding time window. The influence of pruning on the identification system depends on the network architecture at pruning time and the selected weight to be deleted. As the architecture of the model is changed drastically during the identification and pruning process, it is suggested to adapt the pruning interval online. Two system identification examples show the architecture selection ability of the proposed pruning approach.

***Keywords*—**System identification, dynamic neural network, recurrent neural network, GDNN, optimization, Levenberg Marquardt, real time recurrent learning, network pruning, quasi-online learning.

I. Introduction

IN static neural networks the output is directly calculated from the input through forward connections. Dynamic neural networks include delay lines between the layers. So the output depends also on previous inputs and/or previous states of the network. This paper uses dynamic neural networks in which all layers have feedback connections with several time delays, see Fig. 1. The structure of this neural network model can be choosen very general, thus we call it general dynamic neural networks (GDNN).

The weights of the GDNN are trained with the Levenberg-Marquardt (LM) algorithm. LM-training normally is offline. That means all training patterns have to be available before training is started. In order to get an online training algorithm we use a sliding time window, that includes the information of the last Q time steps, see Fig. 2. With the last Q errors the Jacobian matrix is calculated quasi-online. If the time window is large enough, it can be assumed that the information content of the training data is constant. The pruning algorithm suggested in this paper uses the same training data from this sliding time window to calculate the exact saliency for every weight. The saliency is defined as the change in the output error if a special weight is deleted.

In the system identification process we want to find a sufficient model for an unknown plant. As we do not know the structure of the plant we start with a larger network architecture than necessary. During the optimization process irrelevant weights are deleted. There are several pruning algorithms [1]-[4]. Two well known methods in static neural networks are optimal brain damage (OBD) [1] and optimal brain surgeon (OBS) [2]. The OBD method approximates the saliency with the pivot elements of the Hessian without retraining after the pruning step. The OBS uses the complete Hessian information to approximate the saliency, which is regarded as a continuation of the OBD method. After an OBS pruning step the remaining weights are retrained. The exact pruning approach in this paper calculates the saliencies exactly and uses a retraining step like in OBS pruning. The structure of the GDNN is changed drastically during the pruning and identification process. So the influence of one pruning step to the identification system can be quite different. The deletion of one weight in a very large neural network usually does not influence the identification system very much and the error after pruning can be reduced quite fast. Whereas the deletion of one weight in a small neural network can influence the system extremely leading to huge model errors. In the latter case the identification system needs more time to cope with the structural changes in the GDNN. In this paper the time between two pruning steps, the so-called pruning interval, is adapted. The time adaption makes use of a scaling algorithm by evaluating the mean error between successive pruning steps. A too short pruning interval leads to high mean errors.
Reducing the model size has a lot of advantages:

- the generalization is improved
- the training speed is getting faster
- larger networks have better local and global minima, the goal is to keep the low cost function during pruning
- since the optimization problem after a pruning step is changed it is possible to overcome a local minimum

The next section presents the recurrent neural network used for system identification. Administration matrices are introduced to manage the pruning process. Section III deals with parameter optimization and the quasi-online approach used throughout this paper. In Section IV the exact pruning algorithm with adaptive pruning interval for architecture selection in GDNNs is suggested. Identification examples are shown in section V. Finally, in Section VI we summarize the results.

II. General Dynamic Neural Network (GDNN)

De Jesus described in his doctoral theses [7] a broad class of dynamic networks, he called the framework layered digital dynamic network (LDDN). The sophisticated formulations and notations of the LDDN allow an efficient computation of the

Christian Endisch, Christoph Hackl and Dierk Schröder are with the Institute for Electrical Drive Systems, Technical University of Munich, Arcisstraße 21, 80333 München, Germany, e-mail: christian.endisch@tum.de.

Jacobian matrix using real-time recurrent learning (RTRL). Therefore we follow these conventions suggested by De Jesus. In [5]-[8] the optimal network topology is assumed to be known. In this paper the network topology is unknown and so we choose an oversized network for identification. In GDNN all feedback connections exist with a complete tapped delay line (from a first-order time delay element z^{-1} up to the maximum order time delay element $z^{-d_{max}}$). The output of a tapped delay line (TDL) is a vector containing delayed values of the TDL input. Also the network inputs have a TDL. Fig. 1 shows a three-layer GDNN. The simulation equation for layer m is

$$\begin{aligned}\underline{n}^m(t) = &\sum_{l\in L_m^f}\sum_{d\in DL^{m,l}} \underset{\sim}{LW}^{m,l}(d)\cdot \underline{a}^l(t-d)+\\ &\sum_{l\in I_m}\sum_{d\in DI^{m,l}} \underset{\sim}{IW}^{m,l}(d)\cdot \underline{p}^l(t-d)+\underline{b}^m\end{aligned} \tag{1}$$

$\underline{n}^m(t)$ is the summation output of layer m, $\underline{p}^l(t)$ is the l-th input to the network, $\underline{IW}^{m,l}$ is the input weight matrix between input l and layer m, $\underline{LW}^{m,l}$ is the layer weight matrix between layer l and layer m, $\underline{b}^m$ is the bias vector of layer m, $DL^{m,l}$ is the set of all delays in the tapped delay line between layer l and layer m, $DI^{m,l}$ is the set of all input delays in the tapped delay line between input l and layer m, I_m is the set of indices of input vectors that connect to layer m, L_m^f is the set of indices of layers that directly connect forward to layer m. The output of layer m is

$$\underline{a}^m(t) = \underline{f}^m(\underline{n}^m(t)) \tag{2}$$

where $\underline{f}^m(\cdot)$ are nonlinear transfer functions. In this paper we use $tanh$-functions in the hidden layers and linear transfer functions in the output layer. At each point of time the equations (1) and (2) are iterated forward through the layers. Time is incremented from $t = 1$ to $t = Q$. (See [7] for a full description of the notation used here). In Fig. 1 the information under the matrix-boxes and the information under the arrows denote the dimensions. R^m and S^m respectively indicate the dimension of the input and the number of neurons in layer m. $\underline{\hat{y}}$ is the output of the GDNN. During the identification process the optimal network architecture should be found. Administration matrices show us, which weights are valid.

A. Administration Matrices

The layer weight administration matrices $\underset{\sim}{\mathcal{AL}}^{m,l}(d)$ have the same dimensions as the layer weight matrices $\underset{\sim}{LW}^{m,l}(d)$ of the GDNN. The input weight administration matrices $\underset{\sim}{\mathcal{AI}}^{m,l}(d)$ have the same dimensions as the input weight matrices $\underset{\sim}{IW}^{m,l}(d)$. The bias weight administration vectors $\underline{\mathcal{Ab}}^m$ have the same dimensions as the bias weight vectors $\underline{b}^m$. The elements in the administration matrices indicates which weights are valid or not, e.g. if the layer weight $lw_{k,i}^{m,l}(d) = [\underset{\sim}{LW}^{m,l}(d)]_{k,i}$ from neuron i of layer l to neuron k of layer m with a dth-order time-delay is valid, then $\left[\underset{\sim}{\mathcal{AL}}^{m,l}(d)\right]_{k,i} = \alpha l_{k,i}^{m,l}(d) = 1$. If the element in the administration matrix equals to zero the corresponding weight has no influence on the GDNN. With these definitions the kth output of layer m can be computed by

$$\begin{aligned}n_k^m(t) = &\sum_{l\in L_m^f}\sum_{d\in DL^{m,l}}\left(\sum_{i=1}^{S^l} lw_{k,i}^{m,l}(d)\cdot\alpha l_{k,i}^{m,l}(d)\cdot a_i^l(t-d)\right)\\ &+\sum_{l\in I_m}\sum_{d\in DI^{m,l}}\left(\sum_{i=1}^{R^l} iw_{k,i}^{m,l}(d)\cdot\alpha i_{k,i}^{m,l}(d)\cdot p_i^l(t-d)\right)\\ &+b_k^m\cdot\alpha b_k^m\\ a_k^m(t) = &f_k^m(n_k^m(t))\end{aligned} \tag{3}$$

where S^l is the number of neurons in layer l and R^l is the dimension of the lth input. Table I shows an example of administration matrices for a three-layer GDNN with 3 Neurons in each hidden layer and $d_{max} = 2$ at the beginning of a pruning process. Only the weight $\alpha l_{1,3}^{2,2}(1)$ from neuron 3 of layer 2 to neuron 1 of the same layer 2 with first-order time-delay is deleted. All other weights are valid.

TABLE I
EXAMPLE OF ADMINISTRATION MATRICES FOR A THREE-LAYER GDNN, 3 NEURONS IN THE HIDDEN LAYERS AND $d_{max} = 2$

Layer	Administration Matrices						
1	$\mathcal{AI}^{1,1}(1)$	$\mathcal{AI}^{1,1}(2)$					
	1	1					
	1	1					
	1	1					
	$\mathcal{AL}^{1,1}(1)$	$\mathcal{AL}^{1,1}(2)$	$\mathcal{AL}^{1,2}(1)$	$\mathcal{AL}^{1,2}(2)$	$\mathcal{AL}^{1,3}(1)$	$\mathcal{AL}^{1,3}(2)$	$\mathcal{Ab}^1$
	1 1 1	1 1 1	1 1 1	1 1 1	1	1	1
	1 1 1	1 1 1	1 1 1	1 1 1	1	1	1
	1 1 1	1 1 1	1 1 1	1 1 1	1	1	1
2	$\mathcal{AL}^{2,1}(0)$		$\mathcal{AL}^{2,2}(1)$	$\mathcal{AL}^{2,2}(2)$	$\mathcal{AL}^{2,3}(1)$	$\mathcal{AL}^{2,3}(1)$	$\mathcal{Ab}^2$
	1 1 1		1 1 0	1 1 1	1	1	1
	1 1 1		1 1 1	1 1 1	1	1	1
	1 1 1		1 1 1	1 1 1	1	1	1
3			$\mathcal{AL}^{3,2}(0)$		$\mathcal{AL}^{3,3}(1)$	$\mathcal{AL}^{3,3}(2)$	$\mathcal{Ab}^3$
			1 1 1		1	1	1

B. Implementation

For the simulations throughout this paper the graphical programming language *Simulink (Matlab)* was used. GDNN, Jacobian calculation, optimization algorithm and pruning were implemented as S-function in *C*.

III. Parameter Optimization

First of all a quantitative measure of the network performance has to be defined. In the following we use the squared error

$$\begin{aligned}E(\underline{w}_k) &= \frac{1}{2}\cdot\sum_{q=1}^{Q}(\underline{y}_q-\underline{\hat{y}}_q(\underline{w}_k))^T\cdot(\underline{y}_q-\underline{\hat{y}}_q(\underline{w}_k))\\ &= \frac{1}{2}\cdot\sum_{q=1}^{Q}\underline{e}_q^T(\underline{w}_k)\cdot\underline{e}_q(\underline{w}_k)\end{aligned} \tag{4}$$

where q denotes the pattern in the training set, $\underline{y}_q$ and $\underline{\hat{y}}_q(\underline{w}_k)$ are, respectively, the desired target and actual model output

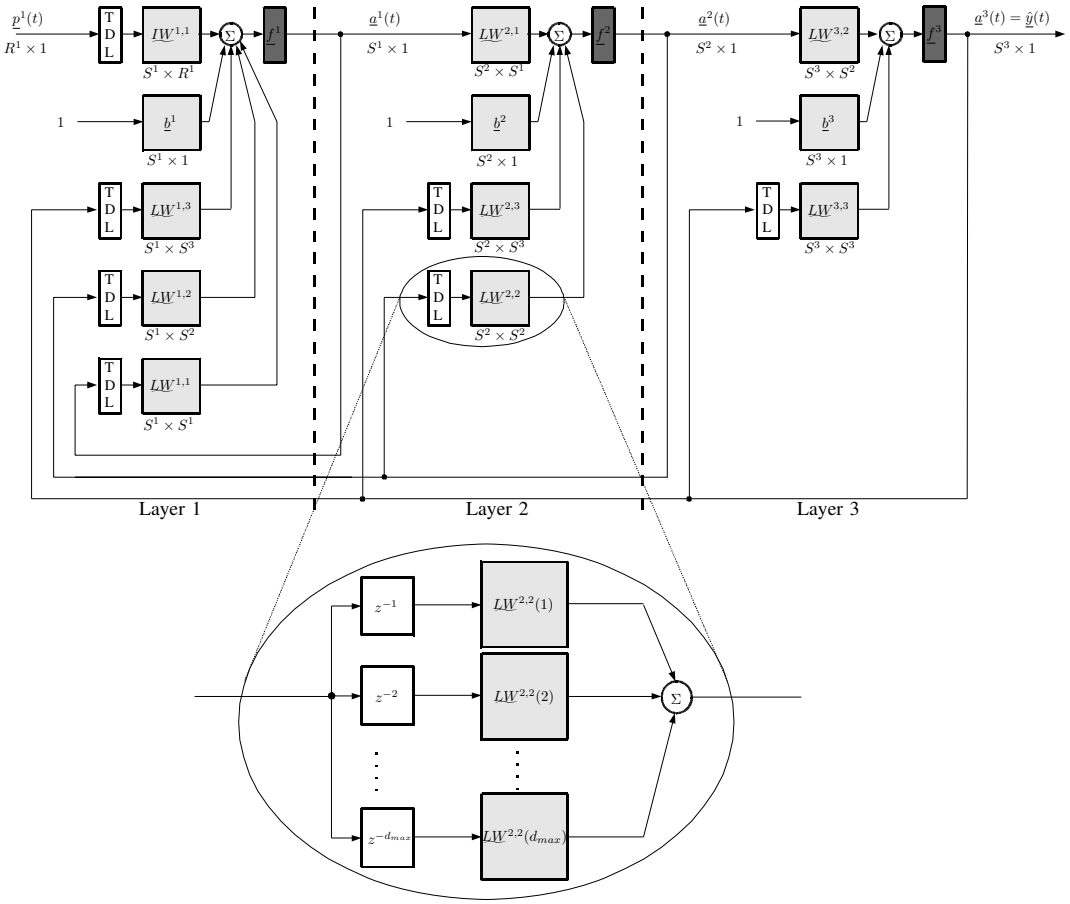

Fig. 1. Three-layer GDNN (two hidden layers)

on the qth pattern. The vector $\underline{w}_k$ is composed of all weights in the GDNN. The cost function $E(\underline{w}_k)$ is small if the training (and pruning) process performs well and large if the it performs poorly. The cost function forms an error surface in a $(n+1)$-dimensional space, where n is equal to the number of weights in the GDNN. In the next step this space has to be searched in order to reduce the cost function.

A. Levenberg-Marquardt Algorithm

All Newton methods are based on the second-order Taylor series expansion about the old weight vector $\underline{w}_k$:

$$\begin{aligned} E(\underline{w}_{k+1}) &= E(\underline{w}_k + \Delta\underline{w}_k) \\ &= E(\underline{w}_k) + \underline{g}_k^T \cdot \Delta\underline{w}_k + \frac{1}{2} \cdot \Delta\underline{w}_k^T \cdot \underline{\underline{H}}_k \cdot \Delta\underline{w}_k \end{aligned} \tag{5}$$

If a minimum on the error surface is found, the gradient of the expansion (5) with respect to $\Delta\underline{w}_k$ is zero:

$$\nabla E(\underline{w}_{k+1}) = \underline{g}_k + \underline{\underline{H}}_k \cdot \Delta\underline{w}_k = 0 \tag{6}$$

Solving (6) for $\Delta\underline{w}_k$ gives the Newton method

$$\begin{aligned} \Delta\underline{w}_k &= -\underline{\underline{H}}_k^{-1} \cdot \underline{g}_k^T \\ \underline{w}_{k+1} &= \underline{w}_k - \underline{\underline{H}}_k^{-1} \cdot \underline{g}_k \end{aligned} \tag{7}$$

The vector $-\underline{\underline{H}}_k^{-1} \cdot \underline{g}_k^T$ is known as the Newton direction, which is a descent direction, if the Hessian matrix $\underline{\underline{H}}_k$ is positive definite. There are several difficulties with the direct application of Newton's method. One problem is that the optimization step may move to a maximum or saddle point if the Hessian is not positive definite and the algorithm could become unstable. There are two possibilities to solve this problem. Either the algorithm uses a line search routine (e.g. Quasi-Newton) or the algorithm uses a scaling factor (e.g. Levenberg Marquardt).

The direct evaluation of the Hessian matrix is computationally demanding. Hence the Quasi-Newton approach (e.g. BFGS formula) builds up an increasingly accurate term for the inverse Hessian matrix iteratively, using first derivatives of the cost function only. The Gauss-Newton and Levenberg-Marquardt approach approximate the Hessian matrix by [9]

$$\underline{\underline{H}}_k \approx \underline{\underline{J}}^T(\underline{w}_k) \cdot \underline{\underline{J}}(\underline{w}_k) \tag{8}$$

and it can be shown that

$$\underline{g}_k = \underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{e}(\underline{w}_k) \tag{9}$$

where $\underline{\mathcal{J}}(\underline{w}_k)$ is the Jacobian matrix

$$\underline{\mathcal{J}}(\underline{w}_k) = \begin{bmatrix} \frac{\partial e_1(\underline{w}_k)}{\partial w_1} & \frac{\partial e_1(\underline{w}_k)}{\partial w_2} & \cdots & \frac{\partial e_1(\underline{w}_k)}{\partial w_n} \\ \frac{\partial e_2(\underline{w}_k)}{\partial w_1} & \frac{\partial e_2(\underline{w}_k)}{\partial w_2} & \cdots & \frac{\partial e_2(\underline{w}_k)}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_Q(\underline{w}_k)}{\partial w_1} & \frac{\partial e_Q(\underline{w}_k)}{\partial w_2} & \cdots & \frac{\partial e_Q(\underline{w}_k)}{\partial w_n} \end{bmatrix} \tag{10}$$

which includes first derivatives only. n is the number of all weights in the neural network and Q is the number of time steps evaluated, see subsection III-B.

With (7), (8) and (9) the Gauss-Newton method can be written as

$$\underline{w}_{k+1} = \underline{w}_k - \left[\underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{\mathcal{J}}(\underline{w}_k)\right]^{-1} \cdot \underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{e}(\underline{w}_k) \tag{11}$$

Now the Levenberg-Marquardt (LM) method can be expressed with the scaling factor μ_k

$$\underline{w}_{k+1} = \underline{w}_k - \left[\underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{\mathcal{J}}(\underline{w}_k) + \mu_k \cdot \underline{\mathcal{I}}\right]^{-1} \cdot \underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{e}(\underline{w}_k) \tag{12}$$

where $\underline{\mathcal{I}}$ is the identity matrix. As the LM algorithm is the best optimization method for small and moderate networks (up to a few hundred weights), this algorithm is used for all simulations in this paper. In the following subsections the calculation of the Jacobian matrix $\underline{\mathcal{J}}(\underline{w}_k)$ and the creation of the error vector $\underline{e}(\underline{w}_k)$ are considered.

B. Quasi-Online Learning

The quasi-online learning approach is shown in Fig. 2. For every optimization step the last Q errors are used for the Jacobian calculation. It is like a time window that slides along the time axis. In every time step the eldest training pattern drops out of the time window. It is assumed, that the training data of the time window describes the optimization problem definitely and so the error surface is constant. The data content of the time window is used for parameter optimization. With this simple method we are able to implement the LM algorithm online. As this quasi-online learning method works surprisingly well, it is not necessary to use a recurrent approach like [10]. For the simulations in this paper the window size is set to 250 steps (using a sampling time of $0.01sec$), but its size can be varied from 10 to 1000 or even more. A change in window size does not have too much influence, as the matrix $\underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{\mathcal{J}}(\underline{w}_k)$ (wich have to be inverted) has $n \times n$ elements.

C. Jacobian Calculations

To create the Jacobian matrix, the derivatives of the errors have to be computed, see Eq. (10). The GDNN has feedback elements and internal delays, so that the Jacobian cannot be calculated by the standard backpropagation algorithm. There are two general approaches to calculate the Jacobian matrix for dynamic systems: By backpropagation-through-time (BPTT) [11] or by real-time recurrent learning (RTRL) [12]. For Jacobian calculations the RTRL algorithm is more efficient than the BPTT algorithm [8]. According to this, the RTRL algorithm is used in this paper. Therefore we make use of the developed formulas of the layered digital dynamic network. The interested reader is referred to [5]-[8] for further details.

IV. Architecture Selection

Pruning algorithms [1]–[4] have initially been designed for static neural networks to obtain good generalization. Due to the quasi-online approach III-B we do not have overtraining problems. This paper is focused on the question: Is it possible to find the optimal structure of a nonlinear dynamic system? In the following subsection we present an exact pruning algorithm for architecture selection in GDNN. In Subsection IV-B the algorithm is improved by an adaptive pruning interval.

A. Exact Network Pruning in GDNN

The goal of network pruning is to set one of the GDNN weights to zero while the cost function given by Eq. (4) is minimized. This particular weight is denoted as $w_{k,z}$. The optimization problem is constrained, so we have to construct a Lagrangian operator. Solving this the optimum change of the weights is [2]

$$\Delta\underline{w}_k = -\frac{w_{k,z}}{\left[\underline{H}_k^{-1}\right]_{z,z}} \cdot \underline{H}_k^{-1} \cdot \underline{i}_z \tag{13}$$

As the LM training algorithm already approximates the Hessian by the Jacobian, this information can be used. With (8) and (12) the optimum change in weights can be approximated by

$$\Delta\underline{w}_k = -\frac{w_{k,z} \cdot (\underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{\mathcal{J}}(\underline{w}_k) + \mu_k \cdot \underline{\mathcal{I}})^{-1} \cdot \underline{i}_z}{[(\underline{\mathcal{J}}^T(\underline{w}_k) \cdot \underline{\mathcal{J}}(\underline{w}_k) + \mu_k \cdot \underline{\mathcal{I}})^{-1}]_{z,z}} \tag{14}$$

The saliency $S_{k,z}$ for the weight $w_{k,z}$ is calculated by exact evaluation of the training data

$$S_{k,z} = \frac{E(\underline{w}_k) - E(\underline{w}_k + \Delta\underline{w}_k)}{(d+1)} \tag{15}$$

where $\Delta\underline{w}_k$ is calculated by Eq. (14). As the final GDNN-model should be as simple as possible, the factor $d+1$ produces a smaller saliency for network weights with higher order of delay. Thus two weights with similar cost functions but different delay orders have a different influence on the pruning process. Bias-weights and forward-weights have no TDL, hence $d = 0$. All other weights have at least one time delay and are therefore more attractive to pruning. The cost function value $E(\underline{w}_k)$ in Eq. (15) is already known and the cost function after the pruning step $E(\underline{w}_k + \Delta\underline{w}_k)$ is calculated by the actual training data within the sliding time window (see subsection III-B).

B. Adaptive Pruning Interval

Every T_p time steps one pruning step is executed. The time constant T_p is called pruning interval. Usually the cost function increases after a pruning step, the optimization problem is changed. The optimization algorithm tries to reduce the error. It can be seen, that deleting one weight in a very large GDNN does not have too much influence on the identification system. There are many redundant weights available. The smaller the

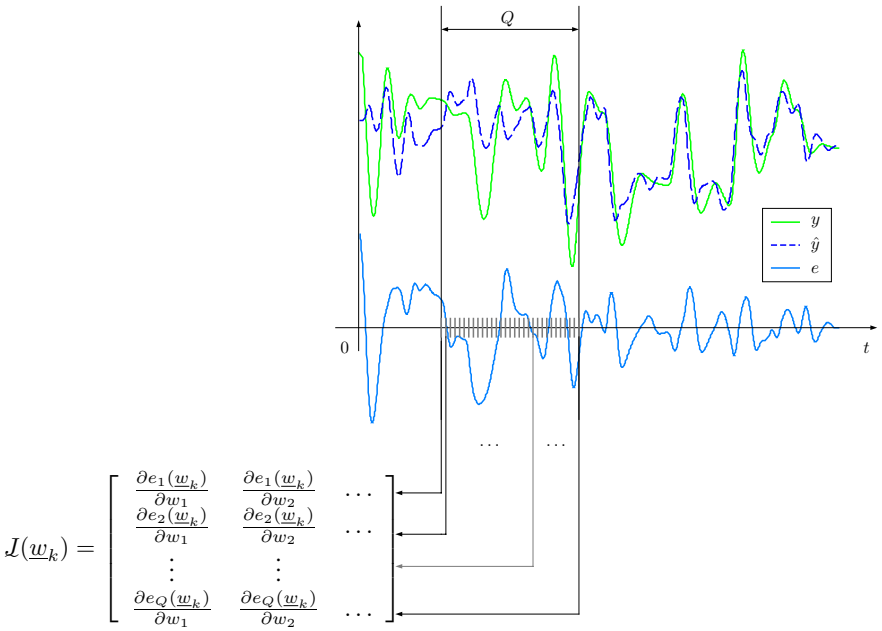

Fig. 2. Quasi-Online Learning

network, the more complex the retraining process after the pruning step. For this reason the pruning process can be improved, if the pruning interval T_p is adapted online. It must be assured, that the optimization algorithm is able to cope with the structural changes in the GDNN-model within the pruning interval T_p.

The cost function value E_{0p} before pruning has to be stored to evaluate the success of the pruning process. Every pruning step starts with the evaluation of the last pruning step. The user defined parameter $\Delta_p > 1$ limits the maximum error increase. The last pruning step is cancelled if

$$E(\underline{w}_k) > \Delta_p \cdot E_{0p} = \Delta_p \cdot E(\underline{w}_{k-T_p-1}) \tag{16}$$

Therefore also the GDNN weights have to be stored before pruning. In Eq. (4) the index q denotes the pattern in the training set within the time window, see Fig. 2. To define the mean error, this relative index has to be substituted:

$$E(\underline{w}_k) = \frac{1}{2} \cdot \sum_{q=0}^{Q-1} (\underline{y}_{k-q} - \underline{\hat{y}}_{k-q}(\underline{w}_k))^T \cdot (\underline{y}_{k-q} - \underline{\hat{y}}_{k-q}(\underline{w}_k)) \tag{17}$$

The mean error E_{mean} between two pruning steps can be computed as

$$E_{mean} = \frac{1}{2T_p} \cdot \sum_{t=k-T_p}^{k} \sum_{q=0}^{Q-1} (\underline{y}_{t-q} - \underline{\hat{y}}_{t-q}(\underline{w}_t))^T \cdot (\underline{y}_{t-q} - \underline{\hat{y}}_{t-q}(\underline{w}_t)) \tag{18}$$

This mean error indicates, if the actual pruning interval T_p is suitable. If the mean error E_{mean} is high, the optimization algorithm cannot cope with the new optimization task within the time T_p, the pruning interval should be increased. If the mean error E_{mean} is small, the pruning process could be sped up. The pruning interval T_p is adapted by the following scaling algorithm:

$$\begin{aligned} &\texttt{if} \quad E_{mean} > \Delta_p \cdot E_{0p} \\ &\qquad T_p = \vartheta \cdot T_p \\ &\texttt{else} \\ &\qquad T_p = \frac{1}{\vartheta} \cdot T_p \end{aligned} \tag{19}$$

with the user defined scaling factor $\vartheta > 1$.

V. Identification

A crucial task in identification with recurrent neural networks is the selection of an appropriate network architecture. How many layers and how many nodes in each layer should the neural network have? Should the layer have feedback connections and what about the taped delay lines between the layers? Should every weight in a layer weight matrix or an input matrix exist? It is very difficult to decide a priori what architecture and what size are adequate for an identification task. There are no general rules for choosing the structure before identification. A common approach is to try different configurations until one is found and works well. This approach is very time consuming if many topologies have to be tested. In this paper we start with a "larger than necessary" network and remove unnecessary parts, so we only require an estimate of what size is "larger than necessary".

In this section two simulation results for plant identification are presented, using the exact pruning algorithm suggested in (14) and (15), with adaptive pruning interval suggested in (18) and (19), Jacobian calculation with real time recurrent learning and the Levenberg Marquardt algorithm (12). The first example to be identified is a simple linear PT_2 system. This example is chosen to test the architecture selection ability of the exact pruning approach suggested in subsection IV-A.

The second example shows the identification of a nonlinear dynamic system with adaptive pruning interval.

A. Excitation Signal

In system identification it is a very important task to use an appropriate excitation signal. A nonlinear dynamic system requires, that all combinations of frequencies and amplitudes (in the system's operating range) are represented in the signal. In this paper an APRBS-signal (Amplitude Modulated Pseudo Random Binary Sequence) was used, uniformly distributed from -1 to +1. For more information see [13].

B. Identification of PT_2 plant

The simple PT_2-plant $\frac{10}{0.1 \cdot s^2+s+100}$ is identified by a three-layer GDNN (with three neurons in the hidden layers and $d_{max} = 2 \Rightarrow n = 93$). This network architecture is obviously larger than necessary. The initial pruning interval is $T_p = 500$, the maximum error increase is set to $\Delta_p = 100$ and the scaling factor ϑ is defined to 1.1. Fig. 3 shows the weight reduction from $n = 93$ to $n = 10$ and the identification error, which is also reduced although the model is becoming smaller. At the end of the pruning process four pruning steps have to be cancelled due to Eq. (16), the error increase is too hight.

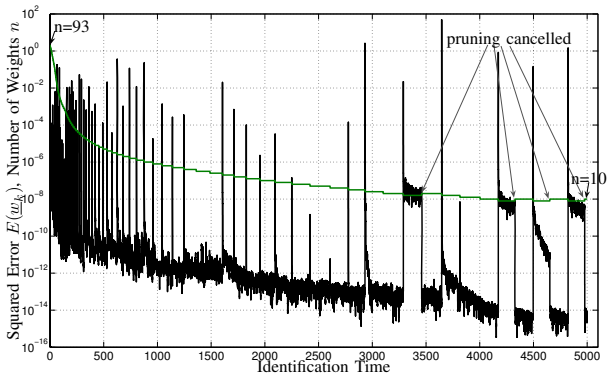

Fig. 3. Squared Error $E(\underline{w}_k)$ and number of weights n for $PT2$-identification example (at the beginning: $n = 93$, at the end: $n = 10$)

To see the online adaption of the pruning interval T_p, in the next Fig. 4 only the first 500 seconds are depicted. The pruning interval decreases for the first 60 seconds, afterwards it increases again.

Table II summarizes the administration matrices after the identification and pruning process.

Most of the weights are deleted. Fig. 5 shows the output of the PT2-plant y (solid gray line), the output of the GDNN-model $\hat{y}$ (thick dashed black line) and the APRBS excitation signal (thin dashed black line) during the first 5 seconds. The output of the GDNN-model follows the output of the plant almost immediately. Fig. 6 shows the identification result and the successful weight reduction of the suggested pruning approach. Each circle in the signal flow chart includes a summing junction and a transfer function. The input and the

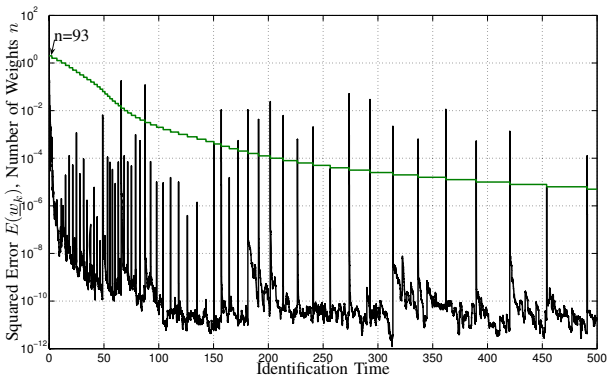

Fig. 4. Squared Error $E(\underline{w}_k)$ and number of weights n for $PT2$-identification example for the first 500 seconds.

TABLE II
ADMINISTRATION MATRICES AFTER PT2-IDENTIFICATION

Layer	Administration Matrices						
1	$\mathcal{AI}^{1,1}(1)$	$\mathcal{AI}^{1,1}(2)$					
	1 1 0	1 1 0					
	$\mathcal{AL}^{1,1}(1)$	$\mathcal{AL}^{1,1}(2)$	$\mathcal{AL}^{1,2}(1)$	$\mathcal{AL}^{1,2}(2)$	$\mathcal{AL}^{1,3}(1)$	$\mathcal{AL}^{1,3}(2)$	$\underline{\mathcal{A}b}^1$
	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0	0 0 0	0 0 0
2	$\mathcal{AL}^{2,1}(0)$		$\mathcal{AL}^{2,2}(1)$	$\mathcal{AL}^{2,2}(2)$	$\mathcal{AL}^{2,3}(1)$	$\mathcal{AL}^{2,3}(1)$	$\underline{\mathcal{A}b}^2$
	1 0 0 0 0 0 0 1 0		0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	0 0 0	0 0 0	0 0 0
3			$\mathcal{AL}^{3,2}(0)$		$\mathcal{AL}^{3,3}(1)$	$\mathcal{AL}^{3,3}(2)$	$\underline{\mathcal{A}b}^3$
			1 0 1		1	1	0

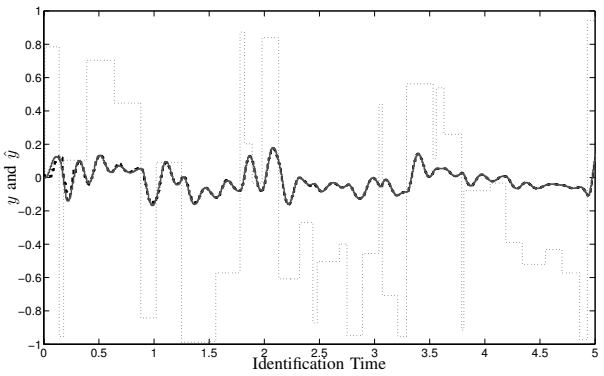

Fig. 5. Output of the PT2-plant y (solid gray line), output of the GDNN-model $\hat{y}$ (thick dashed black line) and APRBS excitation signal (thin dashed black line) for the first 5 seconds.

output of the model has two time delays. If the continuous-time model $\frac{10}{0.1 \cdot s^2+s+100}$ is discretized using zero-order hold [14] with a sample time of $10ms$ the discrete model can be expressed as

$$y(k) = 0.004798 \cdot u(k-1) + 0.00464 \cdot u(k-2) + 1.81 \cdot y(k-1) - 0.9048 \cdot y(k-2) \quad (20)$$

The factors 1.81 and 0.9048 of the difference equation correspond to the feedback weights of the GDNN model. Because

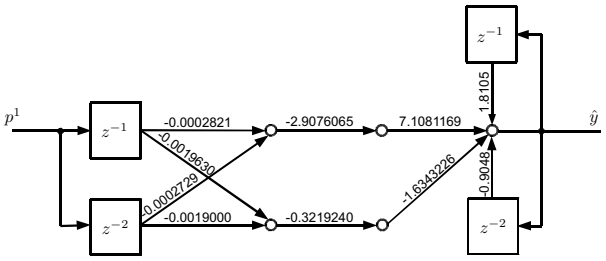

Fig. 6. Identification result: Model of PT2 example

of the very small input weights in the GDNN (see Fig. 6) only the linear region (with gradient 1) of the $tanh$-functions is crucial. Note: Also the other factors of the difference equation (20) correspond exact to the weights of the GDNN (e.g.$-0.0002821 \cdot (-2.9076065) \cdot 7.1081169 - 0.0019630 \cdot (-0.3219240) \cdot (-1.6343226) = 0.0047975$), see Fig. 6.

Note: The final GDNN-model is not unique. The pruning algorithm can find different GDNN-structures for the PT_2 identification example. All these models use the first and second time delay from the input and from the output, like the difference equation of the PT_2-model. Fig. 7 shows another final GDNN-model with $n = 7$ for the same PT_2-identification example. The input weight -0.0027 (see Fig. 6) is quite small

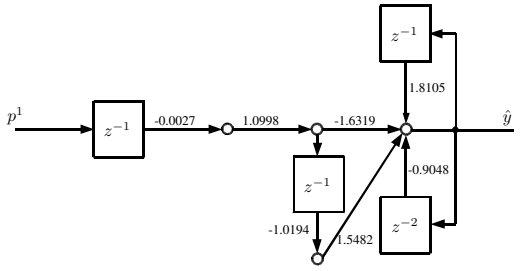

Fig. 7. Identification result: Model of PT2 example

again, to achieve linear behavior. The weights of the final GDNN-model in Fig. 7 correspond to the coefficients of the difference equation (20). The different final GDNN-models have the same input-output behavior.

The LM-scaling factor μ decreases during system identification, this is illustrated in Fig. 8. The developing of μ looks similar to the model error in Fig. 3.

The adaptive pruning interval in the PT2 simulation example assures, that the optimization algorithm can cope with the structural changes. The identification error is reduced, although the number of weights is decreased drastically. If the pruning interval is constant and too small ($T_p = 10$, the other settings are not changed), the optimization algorithm is not able to retrain the remaining weights after pruning. The identification error increases during the pruning process, see Fig. 9.

C. Identification of a nonlinear plant

The first example is chosen to underline the weight reduction ability of the suggested pruning algorithm. In this

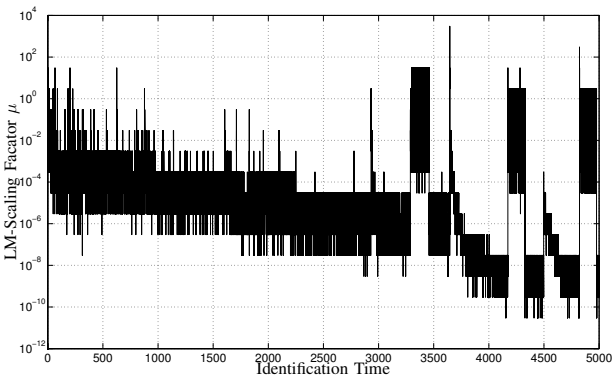

Fig. 8. Decrease in LM-Scaling Factor μ during system identification

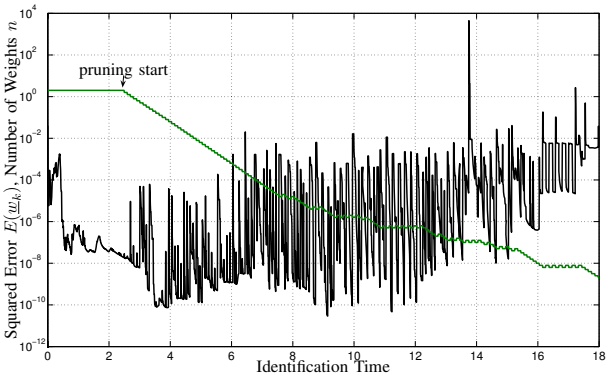

Fig. 9. Constant pruning interval is too small: Squared Error $E(\underline{w}_k)$ and number of weights n for $PT2$-identification example

subsection a nonlinear dynamic system presented by Narendra [15] is considered:

$$y(k) = \frac{y(k-1) \cdot y(k-2) \cdot y(k-3) \cdot u(k-2) \cdot [y(k-3)-1] + u(k-1)}{1 + y^2(k-2) + y^2(k-3)}$$

For this identification example a three-layer GDNN (with four neurons in the hidden layers and $d_{max} = 3 \Rightarrow n = 212$) is used. The initial pruning interval is $T_p = 100$, the maximum error increase is set to $\Delta_p = 20$ and the scaling factor ϑ is defined to 1.1. Fig. 10 shows the weight reduction from $n = 212$ to $n = 57$ and the identification error. The pruning process is stopped after 182 seconds due to the abrupt rise of the cost function $E(\underline{w}_k)$. The last pruning step is cancelled. Fig. 11 shows the output of the nonlinear plant y (solid gray line), the output of GDNN-model $\hat{y}$ (thick dashed black line) and the APRBS excitation signal (thin dashed black line) during the first 2.5 seconds.

VI. Conclusion

In this paper it is shown, that network pruning not only works in static neural networks but can also be applied to dynamic neural networks. For this an exact pruning algorithm with adaptive pruning interval is suggested. The weights to be pruned are found by direct evaluation of the training data

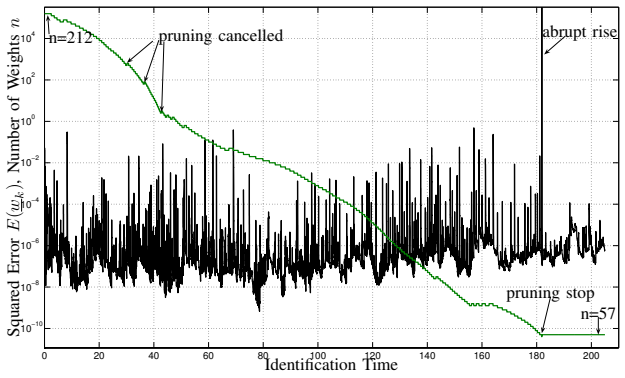

Fig. 10. Squared Error $E(\underline{w}_k)$ and number of weights n for nonlinear identification example (at the beginning: $n = 212$, at the end: $n = 57$)

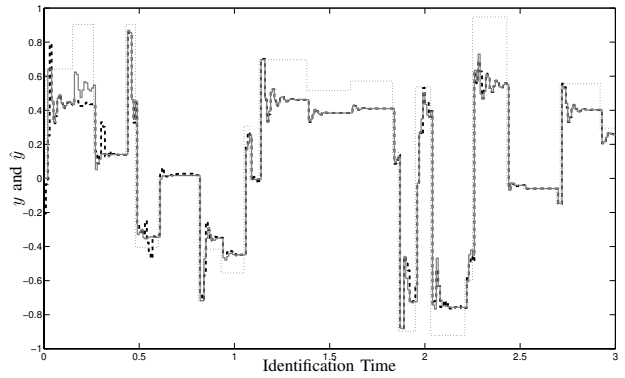

Fig. 11. Output of the nonlinear plant y (solid gray line), output of GDNN-model $\hat{y}$ (thick dashed black line) and APRBS excitation signal (thin dashed black line) for the first 2.5 seconds.

within a sliding time window. Weights with higher order of delay are preferred to pruning. The pruning interval is adapted by a scaling algorithm, as the architecture of the GDNN-model is changed drastically during the identification process. The scaling algorithm uses the mean error between two pruning steps. The mean error indicates, if the pruning interval is suitable. The adaptive pruning interval assures, that the optimization algorithm is able to cope with the structural changes in the GDNN-model within the pruning interval. In every pruning step the success of the last pruning step is checked. If the increase of the error is too high, the last pruning step is cancelled. For this revision, the weights have to be stored in every pruning step.

The Jacobian matrix for the optimization algorithm is calculated quasi-online with RTRL. Quasi-online learning uses a time window for parameter optimization. With this approach the LM algorithm can be applied online. To manage the pruning process with GDNNs, adminstration matrices are introduced. These matrices show us the actual GDNN and indicate, which weights are already deleted. The suggested algorithm finds accurate models of low order. A simple linear system identification example is chosen to show the weight reduction ability. The final GDNN model is not unique. The input weights of the final GDNN models are very small to get an almost linear behavior of the models in the system's operating range. In addition to that the weights of the GDNN models correspond very closely to the coefficients of the according difference equation. Also the nonlinear dynamic system example is identified accurately, whereas the number of weights is reduced drastically.

REFERENCES

[1] Y. Le Cun, J. S. Denker, S. A. Solla, "Optimal Brain Damage," in D. S. Touretzky, "Advances in Neural Information Processing Systems," Morgan Kaufmann, 1990, pp. 598–605.
[2] B. Hassibi, D. G. Stork, G. J. Wolff, "Optimal Brain Surgeon and General Network Pruning," IEEE International Conference on Neural Networks, vol. 1, pp. 293–299, April 1993.
[3] R. Reed, "Pruning Algorithms – A Survey," IEEE Transactions on Neural Networks, vol. 4, no. 5, pp. 740–747, September 1993.
[4] M. Attik, L. Bougrain, F. Alexandre, "Optimal Brain Surgeon Variants For Feature Selection," in IEEE Proceedings of the International Joint Conference on Neural Networks, pp. 1371–1374, 2004.
[5] O. De Jesús, M. Hagan, "Backpropagation Algorithms Through Time for a General Class of Recurrent Network," IEEE Int. Joint Conf. Neural Network, Washington, 2001, pp. 2638–2643.
[6] O. De Jesús, M. Hagan, "Forward Perturbation Algorithm For a General Class of Recurrent Network," IEEE Int. Joint Conf. Neural Network, Washington, 2001, pp. 2626–2631.
[7] O. De Jesús, "Training General Dynamic Neural Networks," Ph.D. dissertation, Oklahoma State University, Stillwater, OK, 2002.
[8] O. De Jesús, M. Hagan, "Backpropagation Algorithms for a Broad Class of Dynamic Networks," IEEE Transactions on Neural Networks, vol. 18, no. 1, pp. 14–27, January 2007.
[9] M. Hagan, B. M. Mohammed, "Training Feedforward Networks with the Marquardt Algorithm," IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989–993, November 1994.
[10] L.S.H. Ngia, J. Sjöberg, "Efficient Training of Neural Nets for Nonlinear Adaptive Filtering Using a Recursive Levenberg-Marquardt Algorithm," IEEE Transactions on Signal Processing, vol. 48, no. 7, pp. 1915–1927, July 2000.
[11] P.J. Werbos, "Backpropagation Through Time: What it is and how to to it," Proc. IEEE, vol. 78, no. 10, pp. 1550-1560, 1990.
[12] R.J. Williams, D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," Neural Computing, vol. 1, pp. 270–280, 1989.
[13] O. Nelles, Nonlinear System Identification. Berlin Heidelberg New York: Springer-Verlag, 2001.
[14] D. Schröder, Elektrische Antriebe - Regelung von Antriebssystemen. 2nd edn., Berlin Heidelberg New York: Springer-Verlag, 2001.
[15] K. S. Narendra, K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4–27, November 1990.

Christian Endisch received the Dipl. Ing. degree in Electrical Engineering from Technical University of Munich, Germany, in 2003. Since then he is with the Institute for Electrical Drive Systems at Technical University of Munich as teaching and research assistant. His main research interests are optimization algorithms and system identification with neural networks.

Christoph Hackl received the B.S. degree in Electrical Engineering from the Technical University of Munich (TUM) in 1999 in co-operation with the University of Madison, Wisconsin. In August, 2004 he finished his diploma in Electrical Engineering (TUM) and began to research as post-graduate associate at the Institute for Electrical Drive Systems. His main interests are robust, decentralized and adaptive control strategies for uncertain, nonlinear MIMO-systems.

Dierk Schröder received the Dipl. Ing. and Dr. Ing. degrees from the Technical University Darmstadt, Germany, in 1966 and 1969, respectively. He spent 10 years in different positions with ABB. In 1979 he was named Professor and Chairman of the Institute of Electronics and Power Electronics at the University Kaiserslautern. In 1983 he was named Professor and Chairman of the Institute for Electrical Drive Systems at the Technical University Munich, Germany.