University of Würzburg
Institute of Computer Science
Research Report Series

Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications

K. Tutschku

Report No. 118 June 1995

Institute of Computer Science, University of Würzburg
Am Hubland, 97074 Würzburg, Germany
Tel.: +49-931-8885513, Fax: +49-931-8884601
e-mail: tutschku@informatik.uni-wuerzburg.de

Abstract: *This study investigates the properties of artificial recurrent neural networks. Particular attention is paid to the question of how these nets can be applied to the identification and control of non-linear dynamic processes. Since these kind of processes can only insufficiently be modelled by conventional methods, different approaches are required. Neural networks are considered to be useful for this purpose due to their ability to approximate a wide class of continuous functions. Among the numerous network structures, especially the* recurrent multi-layer perceptron (RMLP) *architecture is promising from application point of view. This network architecture has the wellknown properties of multi layer perceptrons and moreover these nets have the ability to incorporate temporal behavior. Departing from the original process description the applicability of RMLPs is investigated and different learning algorithms for this network class are outlined. Furthermore, besides the conventional algorithms, like* Back-propagation through time, Real-Time recurrent learning (RTRL) *and* Dynamic Back-propagation, *a more sophisticated training method which uses second order information, the* Global Extended Kalman Filter (GEKF) *is introduced. Finally, three applications of RMLPs in the environment of automotive and telecommunication systems are discussed.*

1 Introduction

Identification with regard to the modeling of processes and the control of them is an important challenge for systems engineers. For example not only does a simple heating of a warm water reservoir require a sophisticated control unit, but also complex systems like cars need to perform a multiplicity of control functions. The modeling of such control systems is a key issue for engineers and determines the functionality of the product and its success.

Conventional modeling techniques are based on linear system theory. They use linear algebra and linear ordinary differential equations to describe the processes. Methods for the adaptive identification of unknown process parameters of linear time-invariant processes are well known, [NP90]. However, applying these techniques to real-world applications very often fail, since most of the processes reveal a nonlinear and dynamic behaviour. Therefore different modeling techniques have to be developed. So far, only few results for nonlinear systems exist and they can only be used on a system-by-system base.

This is the starting-point for the use of artificial neural networks in *identification* and *control* of nonlinear systems. It has been shown, that neural networks are able to model complex nonlinear time-variant processes adaptively. The solutions they obtained are at least comparable to those of the methods established so far, [NP90, FPT90, CBG90, PCA94]. Therefore neural networks can be seen as a versatile, equally powerful tool. Nevertheless there are still many open questions on using neural nets for identification and control: which network architecture is appropriate for a specific task and how has the network to be trained? These two questions are addressed in this study.

The paper is organized as follows. In Section 2, basic concepts of process modeling are introduced and some specific terms used in the paper are defined. In Section 3, the requirements of neural networks that arise from process modeling are identified and a powerful neural network architecture, the *recurrent multi layer perceptron (RMLP)* is described in detail. In Section 4, three well-established learning rules using pure gradient descent methods are discussed. In this Section, especially the evolution and the dependencies of the learnings rules are stressed. Then, Section 5 describes a more sophisticated method to determine unknown system parameters, the *Gobal extended kalman filter*. It is shown how this method can be applied to the training of recurrent multi layer perceptrons. Finally, in Section 6, three applications of recurrent multi layer perceptrons in automotive and telecommunication systems are discussed.

2 Basic Concepts of Process Modeling

Before designing an identification or a controller network, it is essential to think about the system, that will be approximated by the neural network. To characterize the different processes, *control theory* has developed a useful way to describe them, [Gee89, NvdS90, FKK92]. This Section gives, first of all, an short introduction into process modeling, defines some specific terms used in the text, and then shows how neural networks can be used for identification and control.

2.1 Characterization and Representation of Processes

Characterizing a system means to identify all the various physical factors that influence the process behaviour. Hence, one first needs to determine the *static* parameters and then the *dynamic* factors which conduct the system.

In the context of process modeling, the adjective *dynamic* usually indicates that a pro-

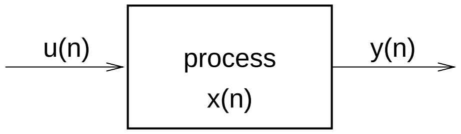


Figure 1: process characterization

cess or a variable reveals a *temporal dependent behaviour*. Therefore *dynamic processes* can be classified into *continuous time* systems, denoted in this text by the index t, and *discrete time* systems, marked by the index n. Since digital micro controller sample systems usually at discrete time intervals, mainly discrete time systems are under consideration in this study. Nevertheless, the characterization for both kind of systems are provided here. A typical discrete time system is depicted in Figure 1. The variable $u(n)$ represents the *external input* to the system at time instant n, the *system state* is denoted by $x(n)$ and the *observable output* of the system is $y(n)$. The variables can either be scalars or vectors. In the case of only a scalar input and a single observable output the system is called a *single-input/single-output (SISO)* system; in the vector case the system is of *multiple-input/multiple-output (MIMO)* type.

Time-invariant Linear System

A *time-invariant linear system* is defined in discrete-time using following *linear* equations, cf. [NvdS90]:

$$x(n+1) = Ax(n) + Bu(n) \tag{1a}$$
$$y(n) = Cx(n) + Du(n), \tag{1b}$$

or in the continuous case by linear differential equations:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2a}$$
$$y(t) = Cx(t) + Du(t), \tag{2b}$$

where A, B, C and D are properly dimensioned matrices. The differential equation in (2a) can either be a first order or a higher order *ordinary differential equation (ODE).*

Time-invariant Nonlinear System

Every time-invariant system that can't be described by eqn. (1) or eqn. (2) is defined to be a *nonlinear system*. It is denoted in discrete time by:

$$\begin{aligned} x(n+1) &= f(x(n), u(n)) \qquad (3a) \\ y(n) &= h(x(n), u(n)), \qquad (3b) \end{aligned}$$

respectively in continuous time:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \qquad (4a) \\ y(t) &= h(x(t), u(t)), \qquad (4b) \end{aligned}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions. The function $f(\cdot)$ is the *state transition function* and $h(\cdot)$ is the *observation function.*

Time-variant Nonlinear System

So far, it was assumed that the characteristics of the system doesn't change over time. To model, for example ageing processes, one has to consider *time-variant* systems. Therefore, the time has to be included as an input factor to the system. In the discrete time case, the nonlinear equations becomes:

$$\begin{aligned} x(n+1) &= f(x(n), u(n), n) \qquad (5a) \\ y(n) &= h(x(n), u(n), n), \qquad (5b) \end{aligned}$$

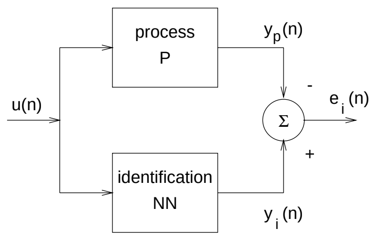
or in continuous time:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), t) \qquad (6a) \\ y &= h(x(t), u(t), t). \qquad (6b) \end{aligned}$$

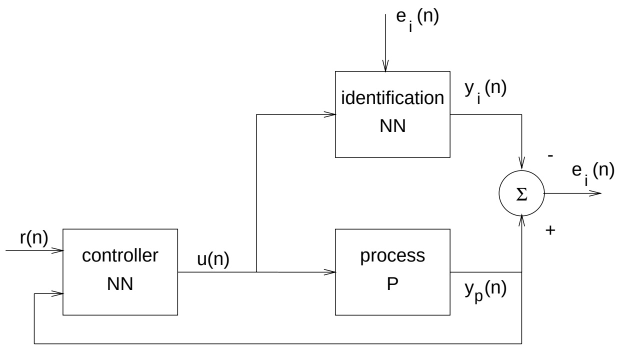
Equations for the linear case are not provided here. To characterize the system, one has to make the matrices A, B, C and D in eqn. (1) and eqn. (2) time dependent.

2.2 Process Identification and Control by Artificial Neural Networks

Employing *artificial neural networks* for identification and control can reduce the expense of control system design considerably. Instead of characterizing the process with complex difference or differential equations and finding the right system parameters, one can exploit the ability of neural networks to learn arbitrary functions, i.e. to learn the process behaviour. The system designer requires a reduced amount of knowledge for describing the process: some rough estimates of the structure and a reasonable set of *input/response patterns* of the actual system are sufficient. An appropriate *learning rule* can then use this information to train the neural net on the predefined task.



(a) identification model



(b) model reference adaptive control

Figure 2: neural network applications in process control

Process Identification

In *process identification* the net has to imitate the actual process, i.e. it has to approximate the process. A typical training configuration of an *identification network* is shown in Figure 2(a). The *actual process* P is parallel to the identification net NN. Both systems get the same external input $u(n)$. In training, the output of the identification net $y_i(n)$ is compared with the process $y_p(n)$ and it stops when the *identification error* $e_i(n)$ drops below a user defined value. The control system designer can benefit from the trained network in two ways: first of all, the net *characterizes* the relevant parameters and the structure of the process and secondly it can be used as a *reference model* for the training of controller networks.

Process Control

Apart from using neural networks as identification nets, they can directly be applied as controllers for processes. Just as in the identification case, one takes advantage of the ability of neural nets to learn an unknown function from a set of examples. For process control, the neural net is trained to imitate the *control function.* A *model reference adaptive control* scenario is depicted in Figure 2(b). The controller neural network translates the *reference signal* $r(n)$ into the appropriate *process control action* $u(n)$. The control action is fed to the actual process and also to the identification neural network which runs parallel. In *on-line* operation, the identification net can be used as a *reference model* for the process. The reference isn't disturbed by *observation noise* like the real process, and therefore it can detect a deviation of the process from the desired behaviour. However, one still needs to decide whether the identification model is right and the process is wrong or vice versa. In *off-line* operation, when training the controller net, the identification network can be used as a substitute for the actual process. Very often the real process is not available, for example the heat exchanger of a power station can not be placed on the desktop of the system designer. Moreover, the identification net can provide estimates for the derivatives of the process output with respect to the trainable controller parameter.

3 Neural Network Architecture

The right choice of the network architecture is crucial for the application of neural nets in process identification and control. Without an appropriate structure, the network couldn't perform it's task sufficiently. However, before determing a well suited architecture, one needs to specify the demands on the neural nets arising from this kind of application.

3.1 Neural Network Requirements

In identification and control, neural nets normally are required have to produce a particular output in response to a specific input sequence, cf. sec. 2.1. Thus, a suitable neural network architecture has to fulfill two main properties:

1) the network structure has to incorporate temporal, i.e. *dynamic*, behaviour,

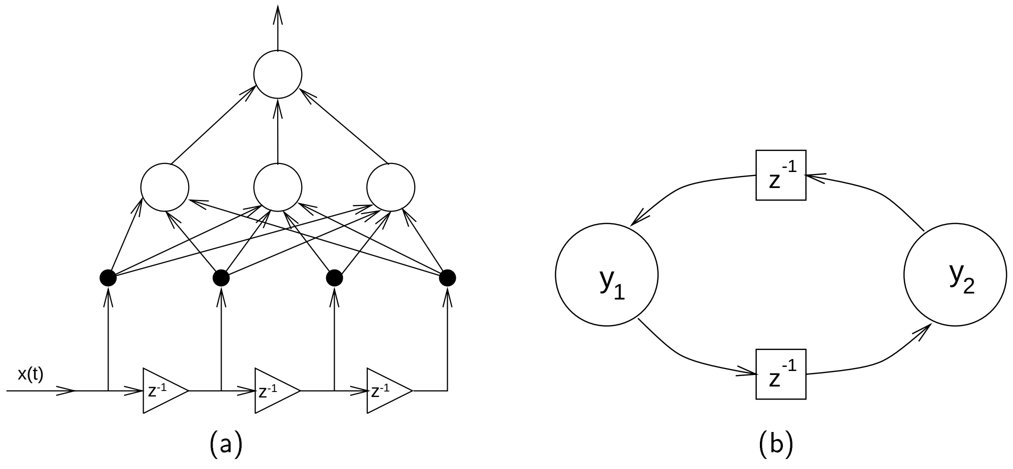
2) the network must be able to approximate arbitrary continuous functions.

Temporal Behavior

A first approach to incorporate time behaviour into neural networks is to use a *shift register* or a *tapped delay line*. This turns the temporal sequence into a spatial pattern on the input layer of a network, cf. Figure 3(a). Tapped delay lines yield reasonable performance on sequence recognition tasks, as long as the sequence is short and of known length. However, this approach has a major drawback. The network sees only a fixed window of the past and therefore can not perform a long-term temporal association. This problem can be solved by either using *context units*, cf. [Jor86, Elm90] or, in general, by introducing *time delayed feedback connections* between the nodes in the network, cf. Figure 3(b). Usually, in time discrete networks the time delay is one time unit, but longer delays are considerable. Due to the *feedback* respectively *recurrent* connections, the net can store the temporal behaviour in it's nodes. It can, therefore "remember" all the *states* it has already gone through and is now able to do temporal association.

Approximation of Continuous Functions

The second demand on a neural network architecture is the ability to learn functions arbitrarily close. Hornik, Stinchcombe and White have shown that the well-known *Stone-Weierstrass-Theorem* can be applied to neural networks, [HSW89]. They stated that standard feed-forward multi layer networks with at least one hidden layer can approximate any *Borel measurable* function at any desired degree of accuracy, provided



a feed-forward net using a tapped delay line

time delayed feedback connections between two nodes

Figure 3: incorporating temporal behaviour in neural networks

sufficiently many hidden units are available. In this sense, feed-forward multi layer networks are a class of *universal approximators.*

Results for Neural Networks

The two previous results taken together, suggest that *recurrent multi-layer networks* should especially be used for the identification and control of processes. This result can be view from a more theoretic point of view. Since multi layer neural networks are a subclass of fully connected nets, the results for feed-forward nets can be applied to some extend to recurrent layered networks. However, one has to be aware that the stability of recurrent networks is not necessarily guaranteed, while purely feed-forward nets always reach a stable output.

3.2 Recurrent Multi Layer Perceptron

A neural network architecture that satisfy the requirements stated in the previous Section, is the generalized version of the *recurrent multi layer perceptron (RMLP).* This architecture was originally presented in [PSY88] and [FPT90]. The generalization has been proposed by Puskorius and Feldkamp and can be found in detail in [PF94].

A general RMLP consists of a sequence of cascaded *subnetworks.* Each subnet consists of layers of nodes. All subnets are interconnected by feed-forward links and no recurrent connections between the subnets are allowed. In the subnetwork, all the layers are again feed-forward connected and only the last layer of the subnet is allowed to have feedback connections between its nodes. As a result, this generalized RMLP structure

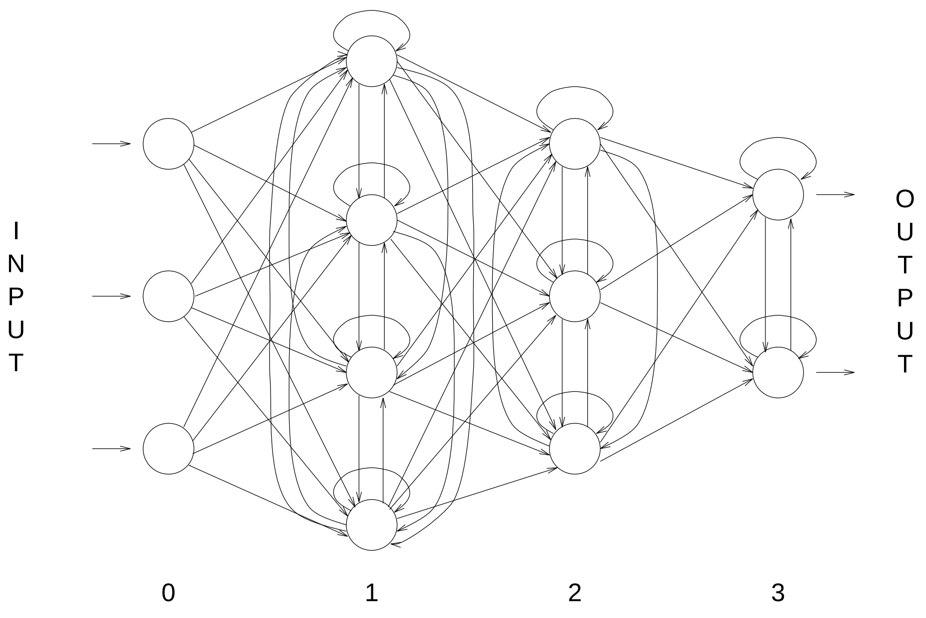


Figure 4: RMLP with two hidden, completely interconnected, recurrent layers

can describe purely feed-forward nets as well as fully recurrent networks, cf. [PF94]. The general RMLP architecture is quite powerful, but for its succesful application, the structure must be more feasible, which in turn, restricts its use to some extend: the subnets are only allowed to have exactly one layer. In this paper, the terms *layer* and *subnetwork* are used as synonyms.

Detailed Network Structure

The investigated discrete time multi layer perceptron with *local* and *lateral*, i.e. layer intern, feedback connections is depicted in Figure 4. Each hidden layer is fully interconnected and linked to the next hidden layer in a feed-forward manner. Thus, each node has trainable recurrent links to itself, to every other node in its layer and to the nodes in the succeeding layer. The recurrent connections are delayed by one time unit. In detail, the layer and the node structure is defined as follow:

a) Layer Structure

The network has a fixed sequence of three distinct types of layers:

1) one input layer

2) one or more recurrent, completely interconnected hidden layer; the recurrent links have one time unit delay and the forward interconnection from one hidden layer to the next is complete.

3) one output layer

Therefore the total number of *subnets* is $\mathcal{L} = 2 +$ number of hidden subnets. The layer numbering starts with 0 and ends at $\mathcal{L} - 1$.

b) Node Structure

The node type of RMLP nets is based on the usual *McCulloch and Pitts model*, cf. [HKP91]. In detail, the *network input* $net_{i,j}$ to node j in layer i is defined as:

$$\mathrm{net}_{i,j} = \sum_{k=1}^{\mathcal{N}_{i-1}} w_{k,j}^{f,i} y_{i-1,k}(n) + \sum_{k=1}^{\mathcal{N}_i} w_{k,j}^{r,i} y_{i,k}(n-1) \tag{7}$$

where $y_{i,j}$ is the activation of node j in layer i, $w_{k,j}^{f,i}$ is a forward weight form node k in layer $(i-1)$ to node j in layer (i), $w_{k,j}^{r,i}$ is a recurrent weight from node k in layer (i) to node j in layer (i) and $\mathcal{N}_i$ is the number of nodes in subnet (i). The time index $(n-1)$ indicates that the feedback is delayed by one time step. The *activation* $y_{i,j}(n)$ of node j in layer i is:

$$y_{i,j}(n) = \sigma(\mathrm{net}_{i,j}(n)), \tag{8}$$

where the operator $\sigma(\cdot)$ describes the *activation function* of the node. Usually, one use either the *logistical activation function*:

$$\sigma(s) = \frac{1}{1-e^{-s}}, \tag{9a}$$

$$\sigma'(s) = \sigma(s)(1-\sigma(s)), \tag{9b}$$

or the *symmetrical transfer function*:

$$\sigma(s) = \tanh s \tag{10a}$$

$$\sigma'(s) = (1+\sigma(s))(1-\sigma(s)). \tag{10b}$$

4 Gradient-Descent Learning Algorithms for RMLP

The training of discrete time recurrent multi layer perceptron is based on essentially the same principle as for non recurrent multi layer perceptrons. The error of the network, that is the difference between the desired output and the actual output, is minimized by using gradient descent on the *cost function* or respectively on the network *error surface*. Nevertheless learning a recurrent network is more complicated than a feed-forward net. Since successive inputs to the network can be highly correlated over a period of time, pure gradient descent methods tend to be slow and often get stuck in an ineffective solutions. *Small learning rates* can reduce the effect but increase learning time. Hence, different learning algorithm for recurrent networks are necessary. Numerous algorithms have been proposed in literature. Three conventional gradient methods are discussed in this Section: *Back-propagation through time (BPTT)*, *Real-time recurrent learning (RTRL)* and *Dynamic Back-propagation (DBP)* respectively *Dynamic Derivatives.*

4.1 Gradient Descent Learning

Generally speaking, the task of a *learning rule* for arbitrary systems is to find a proper set of parameters for the system which allows it to perform near its optimum. In case of neural networks, an *adaptive* training method should find the best set of weights by successive improvements from an arbitrary starting point. To evaluate the learning progress, a quality measure for the network performance is needed. Usually, one defines a *cost function* respectively an *error measure* of the network on a predefined task, i.e. the error on a predefined test set. The *squared error* for one element of the test set at time step n is:

$$E(n) = \frac{1}{2}\sum_k [E_k(n)]^2, \tag{11}$$

with

$$E_k(n) \quad = \quad \begin{cases} y_{d,k}(n) - y_k(n) & \text{if } y_{d,k}(n) \text{ is defined} \\ 0 & \text{otherwise} \end{cases}, \tag{12}$$

where k is an element of the index set of all output units, $y_k(n)$ is the actual output of unit k at time step n and $y_{d,k}(n)$ is the *desired* output of k. The *network error* for a test set respectively a sequence of length N is:

$$E = \sum_{n=0}^{N-1} E(n). \tag{13}$$

The error measures of eqn. (12) respectively eqn. (13) are normally positive and approach zero when the learning rule finds a solution. A reduction of the error measures can be done by the well known gradient descent method. Since $E(n)$ is spanning an *error surface* over the weight space w, the network can be improved by sliding down hill on this surface. Therefore the weight adaption rule is:

$$\Delta w_{ij}(n) \quad = \quad -\eta \frac{\partial E(n)}{\partial w_{ij}} = \eta \sum_k E_k(n) \frac{\partial y_k(n)}{\partial w_{ij}}. \tag{14}$$

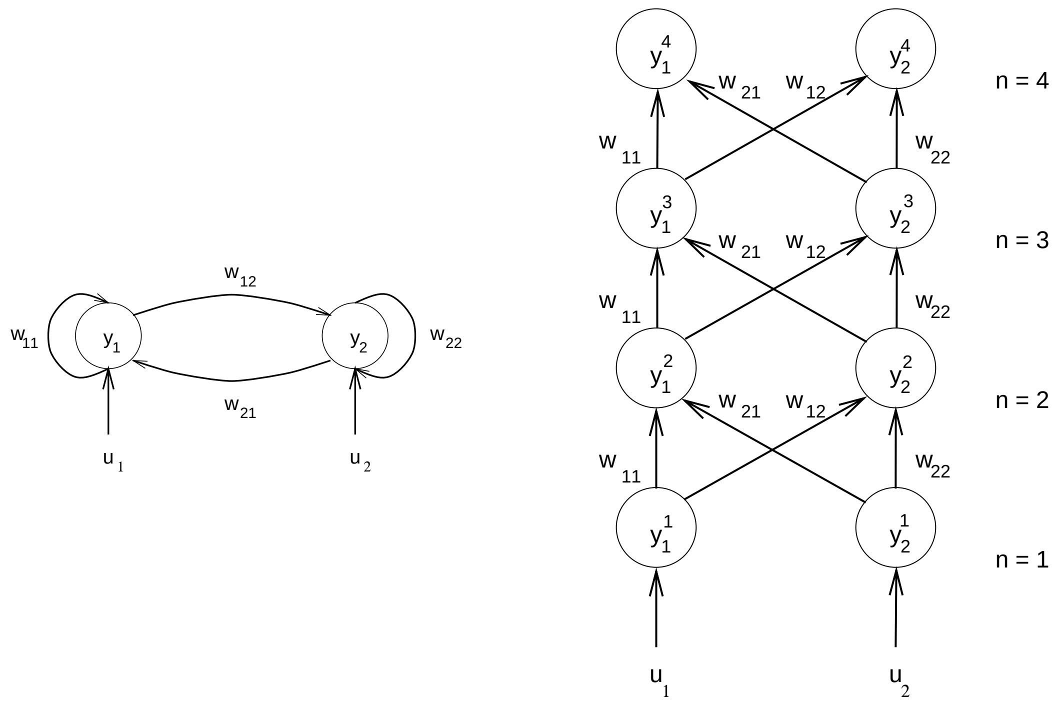
4.2 Back-Propagation Through Time

The *Back-propagation through time* algorithm has been proposed by Rumelhart, Hinton and Williams [RMt86]. It can be applied to synchronous discrete time fully recurrent networks. The update rule for a neuron in this net class is:

$$y_i(n) = \sigma(net_i(n)) = \sigma(\sum_j w_{ji} y_j(n-1) + u_i(n)), \tag{15}$$

where y_i is the state of the unit i at time step n, net_i is the net input and $u(n)$ is the external input, if there is any, to the neuron. The operator $\sigma(\cdot)$ is the activation function, see Section 3.

An example of a fully recurrent network type is shown in Figure 5(a). As mentioned



(a) a recurrent network

(b) an identical behaving feed-forward network (n=4)

Figure 5: Back-propagation through time

above in Section 3.1, the main task of the networks investigated in this study is to produce particular output sequences in response to specific input sequences. If only short input sequences are of interest, a trick can be applied to transform the recurrent network to a purely feed-forward network. For a maximum sequence of length N, the recurrent network is duplicated N times, so that a separate unit Y_i^n represents the state $Y_i(n)$, with $n = 1, \ldots, N$, cf. Figure 5(b). The weights w_{ij} in each layer of the feed-forward net are the same. Thus the two networks in Figure 5 behave identically for N time step. The transformed network can be trained by slightly modified form of the well known Back-propagation algorithm, [RMt86]. The original algorithm would normal change each copy of the weight differently. The modified algorithm therefore has to make sure that all "copies" of each weight w_{ij} must remain identical in the transformed network. This can be obtained by adding all individual increments and than adapt all copies by the same amount. The major drawback of BPTT is that this network structure respectively this training algorithm fails totally for sequences of unknown length.

4.3 Real-Time Recurrent Learning

A learning method for general recurrent networks without duplicating the units has been proposed by Williams and Zipser, [WZ89]. The algorithm allows updating the weights while the sequence is presented. It is therefore called a real time method. The real time ability also constitutes a major advantage of the *Real-time recurrent learning (RTRL)* algorithm over Back-propagation through time: RTRL can handle sequences of of arbitrary length.

Similar to the Back-propagation Through Time algorithm, for a fully recurrent network the state of a unit is governed by:

$$y_i(n) = \sigma(net_i(n)) = \sigma(\sum_j w_{ji} y_j(n-1) + u(n)). \tag{16}$$

The *RTRL* rule is as well a gradient descent method. The last derivative in eqn. (14) is obtained by differentiating eqn. (16) w.r.t. a weight w_{ij}:

$$\frac{\partial y_k(n)}{\partial w_{ij}} = \sigma'(net_i(n))[\delta_{ki} y_j(n-1) + \sum_l w_{kl} \frac{\partial y_k(n-1)}{\partial w_{ij}}] \tag{17}$$

Eqn. (17) constitutes a recursive algorithm for computing the weight change, it relates the derivatives $\partial y_k(t)/\partial w_{ij}$ at time n to those at time $n-1$. A reasonable initial condition for the recursion in eqn. (17) is:

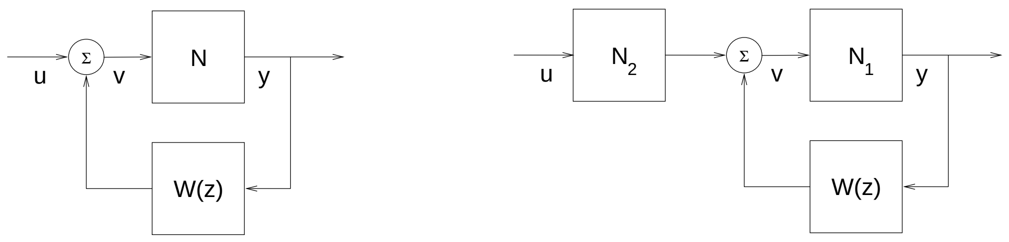
$$\frac{\partial y_k(0)}{\partial w_{ij}} = 0 \tag{18}$$

The eqn. (17) has inherently the real time property of the RTRL algorithm. However this capability doesn't necessarily has to be used. For learning sequences with known length the cost function in eqn. (13) has to be minimized. Therefore all $\Delta w_{ij}(n)$ for $n = 0, \ldots, N-1$ have to be summed up and the weight update is made the total amount after the presentation of the last element of the sequence ($n = N-1$). For learning sequences with arbitrary length one uses the real time property. The weight update can take place after each time step, if the learning rate η has a sufficiently small value, cf. [WZ89].

A useful modification of RTRL is *teacher forcing.* Here instead of feeding back the possible wrong unit value $y_k(n)$, one uses the the target value $y_{d,k}$, if it is available, which is always correct. Of course this must be done after computing the error $E_k(n)$. The teacher forcing procedure keeps the network closer the the desired sequence or trajectory, and usually speeds up convergence. In the gradient calculation of eqn. (17) the derivative $\partial y_k(n)/\partial w_{ij}$ is set to zero whenever unit k is forced to its target value, but it became valid only after the weight adaption.

4.4 Dynamic Back-Propagation and Dynamic Derivatives

As discussed in Section 3.1, for identification and control tasks especially recurrent multi layer perceptrons are well suited and should be applied. In general, these kind of nets



(a) externally recurrent network

(b) cascaded feed-forward and externally recurrent network

Figure 6: Dynamic Back-propagation Architectures

can be trained by the methods described in the previous two subsections. However, using these algorithms means that one does not pay attention to the layered structure of RMLPs. Considering this particular structure in learning algorithms can speed up training, as shown later, since some derivatives can be computed directly and without recursion like in RTRL. An extension of the Back-propagation algorithm is called *Dynamic Back-propagation* and its general application to the RMLP structure, as introduced in Section 3.2 is called *Dynamic Derivatives*.

4.4.1 Dynamic Back-Propagation

The *Dynamic Back-propagation* method was proposed by Narendra and Parthasarathy, [NP90, NP91]. They established this method as a part of a comprehensive but rigorous framework for the training of identification and controller networks. Narendra and Parthasarathy explicitly noted that the controller and the process form a closed-loop feedback system, and that the computation of the derivatives used during training must evolve recursively, just as the process outputs do. Following this line of inquery, they examined first externally recurrent multi layer perceptrons and transferred the results to consecutive series of pure feed-forward and recurrent nets.

Externally Recurrent Back-Propagation Networks

An externally recurrent multi layer perceptron is depicted in Figure 6(a). The purely feed-forward network is denoted by N and the output y of the static net is feed back through the *transfer matrix* $W(z)$.

The multi layer perceptron N represents a static map, whereas the matrix $W(z)$ defines the dynamical properties of the system. For example $W(z)$ can be a diagonal matrix with elements of the form z^{-d_i} (i.e., a delay of d_i units of time), or $W(z) = \sum_{i=1}^{d} \alpha_i z^{-i}$, a finite pulse response of duration d time units.

The output of the system in Figure 6(a) is governed by:

$$y(n+1) = N[u + W(z)y(n)] \tag{19}$$

where u is the external input to the system. The computation of the system output y depends on the weights w_i of network N in two different ways: explicitly on the "current" values and implicitly of "past" values of the parameters w_i, hidden in the feed back value of y. In such a case the interest is in the total derivative of y with respect to w_i. Since equation (19) is of the form:

$$y = f[x(k,w), w], \tag{20}$$

the total derivative of f with respect to w_i can be obtained by the *general chain rule*:

$$\begin{aligned}\frac{\partial f[x(k,w)]}{\partial w_i} = \frac{\bar{\partial} f}{\bar{\partial} w_i} &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial w_i} + \frac{\partial f}{\partial w_i}\frac{\partial w_i}{\partial w_i} \\ &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial w_i} + \frac{\partial f}{\partial w_i}.\end{aligned} \tag{21}$$

The notation $\bar{\partial} f/\bar{\partial} w_i$ is used to denote the total derivative in place of the usual notation d/dw_i to emphasize the vector nature of the parameter w. It also indicates the recursion property of eqn. (21).

Narendra and Parthasarthy, base their argument for *Dynamic Back-Propagation* on the key assumption that feed back transition matrix $W(z)$ is fix and is not allowed to be adapted by the training algorithm. Using this and the result from eqn. (21) applied on eqn. (19), one arrives at:

$$\begin{aligned}\frac{\bar{\partial} y}{\bar{\partial} w_i} &= \frac{\partial N[v]}{\partial v}\frac{\partial v}{\partial w_i} + \frac{\partial N[v]}{\partial w_i} \\ &= \frac{\partial N[v]}{\partial v} W(z) \frac{\bar{\partial} y}{\bar{\partial} w_i} + \frac{\partial N[v]}{\partial w_i}.\end{aligned} \tag{22}$$

In eqn. (22) $\bar{\partial} y/\bar{\partial} w_i$ is a vector and $\partial N[v]/\partial v$ and $\partial N[v]/\bar{\partial} w_i$ are the Jacobian matrix and vector which are evaluated around the the nominal trajectory. The later two components can be computed at every instant by using standard techniques.

Eqn. (22) relates the partial derivative $\bar{\partial} y/\bar{\partial}\theta_i$ recursively to its old values like RTRL it does, cf. sec. 4.3. The result from eqn. (22) can now be used in the weight adaption step, cf. eqn. (14).

As well as in the RTRL case, teacher forcing can be used for externally recurrent nets:

$$y(n+1) = N[u + W(z)y_d(n)], \tag{23}$$

where y_d is the desired output of the system. The network follows the desired trajectory faster. But there is still the problem that only small learning rates η can be used.

Cascaded Feed Forward and Recurrent Multi Layer Perceptrons

The next step is to extend the results from the previous subsection to a cascaded network architecture. Figure 6(b) shows such a structure.

A static, purely feed-forward network N_2 is followed by an externally recurrent multi layer perceptron N_1. The preceeding network N_2 does not effect the computation of the partial derivative of the output y w.r.t. the weights in network N_1. Therefore the same procedure as in the externally recurrent network case can be adopted. However, to compute the partial derivative the output y w.r.t. the weights in network N_2, the following procedure is used. The output of the system in Figure 6(b) is governed by:

$$y = N_1[v] = N_1[N_2[u] + W(z)y], \tag{24}$$

where the operators $N_1[\cdot]$ and $N_2[\cdot]$ describe the static network behaviour and $W(z)$ the dynamic feed back property. The partial derivative of y w.r.t. the weights in N_2 is:

$$\begin{aligned} \frac{\partial y}{\partial w_i} &= \frac{N_1[v]}{\partial v}\frac{\partial v}{\partial w_i} \\ &= \frac{\partial N_1[v]}{\partial v}[\frac{\partial N_2[u]}{\partial w_i} + W(z)\frac{\partial y}{\partial w_i}]. \end{aligned} \tag{25}$$

For the computation of the Jacobian matrices and vectors, again linearization around the operating point is needed.

When the results from eqn. (22) and (25) are combined, one arrives at a method fot computing the partial derivatives of a broad class of cascaded recurrent multi layers perceptrons. The transfer of this method for RMLP architecture of Section 3.2 is described in the next section of this study.

4.4.2 Dynamic Derivatives for RMLP Networks

Using the results obtained by Narendra et. al., the *Dynamic Back-propagation* method has been adapted by Puskarious et. al. to the RMLP architecture [PF91, PF93, PCA94, PF94] as invented by Fernandez et.al. [FPT90], cf. sec. 3.2. The previous restriction, namely that the learning method is not allowed to adapt the feedback connections in training, is now dropped. A recursive algorithm evaluates the derivative of the *subnetworks*[1] output nodes. This method is called *Dynamic Derivatives.*

For the description of the *Dynamic Derivative*, first the network is limited to have only local, but no externally feedback connections from the output to the input layer. Afterwards, this restriction is abandoned and the Dynamic Derivatives are derived when there is output-to-input recurrence.

[1]cf. to the definition of *subnetwork* in sec. 3.2

Dynamic Derivatives for RMLP Networks with no external feedback connections

As described in eqn. (7) and (8) in Section 3.2 the activation of a subnet output node is:

$$y_{i,j}(n) \;=\; \sigma(\sum_{k=1}^{\mathcal{N}_{i-1}} w_{k,j}^{f,i} y_{i-1,k}(n) + \sum_{k=1}^{\mathcal{N}_i} w_{k,j}^{r,i} y_{i,k}(n-1)).$$

The computation of the Dynamic Derivative of the outputs of an RMLP network with respect to its trainable weights can be obtained from three observations:

1) the dynamic derivative of an output node in subnet i with respect to any weight in a *higher* subnet g, thus $g > i$, is zero (i.e. there is no influence from a weight in a "higher" layer on a output node in a "lower" layer).

2) the dynamic derivative of an output node in a subnet with respect to a recurrent weight from the same subnet can be obtained by a generalization of RTRL (cf. with sec.4.3).

3) the dynamic derivative of an output node in the subnet i w.r.t. a weight in a "lower" subnet g, thus $g < i$, is a function of *a)* the dynamic derivative of the outputs of the subnet $(i-1)$, since each output node in subnet $i-1$ has feed-forward connections to every node in subnet i and *b)* of the outputs of the subnet i, since each node in subnet i has feedback connections from every other node within his subnet. Both derivatives are with respect to the same weight in subnet g.

The dynamic derivative of an output node j in subnet i w.r.t. a weight in subnet g, with $g < i$, is found to be:

$$\begin{aligned}\frac{\bar{\partial} y_{i,j}(n)}{\bar{\partial} w_{k,j}^{x,g}} \;&=\; (1-\delta_{g,i}) \sum_{k=1}^{\mathcal{N}_{i-1}} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)} \frac{\bar{\partial} y_{i-1,k}(n)}{\bar{\partial} w_{h,j}^{x,g}} \\ &\quad + \gamma(n) \sum_{k=1}^{\mathcal{N}_i} \frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)} \frac{\bar{\partial} y_{i,k}(n-1)}{\bar{\partial} w_{h,j}^{x,g}} \\ &\quad + \delta_{g,i} \frac{\partial y_{i,k}(n)}{\partial w_{h,j}^{x,g}}\,; \qquad \text{for} \quad g \le i, n \ge 1. \end{aligned} \tag{26}$$

The initial values for the recursion are:

$$\frac{\bar{\partial} y_{i,j}(0)}{\bar{\partial} w_{k,j}^{x,g}} \;=\; 0 \tag{27}$$

The impact of the past dynamic derivatives is conducted by the discount factor $\gamma(n)$. It imposes an exponential decay of the influence. Its value is usually set equal to or less than unity.

The index x specifies the kind of weight connections. A feed-forward link is denoted by 'f' and a feedback connection is marked by an 'r'. The index 'g' describes the layer number in which to node lies where the weight ends. The Operator $\delta_{i,j}$ is the Kronecker delta:

$$\delta_{i,j} = \begin{cases} 0 & \text{for}\, i \neq j \\ 1 & \text{for}\, i = j \end{cases} \tag{28}$$

Since the algorithm is recursive in time and in space, the computed partial derivative $\bar{\partial} y_{i,j}(n)/\bar{\partial} w_{k,j}^{x,g}$ is called *Dynamic Derivative.* The algorithm propagates the derivatives forward in space and in time through the layers. Hence, this method has the real-time property as well as the RTRL algorithm, but also incorporates the spatial dependencies of the nodes.

Computation of the individual components of the dynamic derivative

At a first glance eqn. (26) seems difficult to compute. If one looks closer at the partial derivatives however, one realizes shows, that the gradient can be evaluated very quickly. The different partial derivatives are obtained by the following equations:

1) the partial derivative of an output w.r.t. the output in a preceding layer (i.e. a feed-forward connection) is:

$$\begin{aligned} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)} &= \frac{\partial}{\partial y_{i-1,k}(n)} \sigma(net_{i,j}(n)) \\ &= \sigma'(\mathrm{net}_{i,j}(n)) \cdot w_{k,j}^{f,i} \quad \text{with}\, n \geq 1 \,\text{and}\, i \geq 1 \\ &\overset{(*)}{=} \sigma(\mathrm{net}_{i,j}(n))(1 - \sigma(\mathrm{net}_{i,j}(n))) w_{k,j}^{f,i} \end{aligned} \tag{29}$$

The condition $i \geq 1$ is equivalent with considering only *actual* computing nodes.

2) the partial derivative of an output node w.r.t. to the time delayed output from the same layer (i.e. a recurrent connections) is:

$$\begin{aligned} \frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)} &= \frac{\partial}{\partial y_{i,k}(n-1)} \sigma(net_{i,j}(n)) \\ &= \sigma'(\mathrm{net}_{i,j}(n)) \cdot w_{k,j}^{r,i} \qquad n \geq 1 \\ &\overset{(*)}{=} \sigma(\mathrm{net}_{i,j}(n))(1 - \sigma(\mathrm{net}_{i,j}(n))) w_{k,j}^{r,i} \end{aligned} \tag{30}$$

3) the partial derivative of an output w.r.t. a recurrent weight is:

$$\begin{aligned} \delta_{g,i} \frac{\partial y_{i,j}(n)}{\partial w_{h,j}^{x,g}} &= \begin{cases} 0 & g \neq i \\ \frac{\partial}{\partial w_{h,j}^{r,i}} \sigma(net_{i,j}(n)) & \text{else} \end{cases} \\ &= \begin{cases} 0 & g \neq i \\ \sigma'(net_{i,j}(n)) y_{i,j}(n-1) & \text{else} \end{cases} \\ &\overset{(*)}{=} \begin{cases} 0 & g \neq i \\ \sigma(net_{i,j}(n))(1 - \sigma(net_{i,j}(n))) y_{i,j}(n-1) & \text{else} \end{cases} \end{aligned} \tag{31}$$

(*)when substituted with the logistical transfer function

Especially in this case, one has to pay attention to the fact that the hidden layers of the RMLP architecture investigated in this study are completely connected.

Eqn. (29)–(31) indicate, that the dynamic derivative of a subnetwork's output node, eqn. (26), can be computed in a single sweep through the network. Parallel to the forward propagation of the signal, the partial derivatives can be obtained.

Dynamic Derivative for Externally Recurrent Networks

Now the assumption that there is no external feedback connection between the output and the input layer is dropped. Because of the external recurrence, the dynamic derivative of any subnetwork's output with respect to any weight within an externally recurrent RMLP network is generally nonzero. Therefore eqn. (26) becomes:

$$\begin{aligned}\frac{\bar{\partial} y_{i,j}(n)}{\bar{\partial} w_{k,j}^{x,g}} &= \sum_{k=1}^{\mathcal{N}_{i-1}} \frac{\partial y_{i,j}(n)}{\partial y_{i-1,k}(n)} \frac{\bar{\partial} y_{i-1,k}(n)}{\bar{\partial} w_{h,j}^{x,g}} \\ &\quad + \gamma(n) \sum_{k=1}^{\mathcal{N}_{i}} \frac{\partial y_{i,j}(n)}{\partial y_{i,k}(n-1)} \frac{\bar{\partial} y_{i,k}(n-1)}{\bar{\partial} w_{h,j}^{x,g}} \\ &\quad + \delta_{g,i} \frac{\partial y_{i,j}(n)}{\partial w_{h,j}^{x,g}} \,. \end{aligned} \tag{32}$$

Calculating the Dynamic Derivative for externally recurrent RMLP networks generally requires increased computational resources, since storage and computation of the dynamic derivatives of all recurrent nodes w.r.t. all trainable weights of the network are required.

5 Training Neural Networks with the Global Extended Kalman Filter Algorithm

In identification and control applications, neural networks are used to approximate processes. Therefore it is obvious to model the nets themself as a process and determine the parameters by using *control theory* methods. This procedure seems to be obsolete since the use of conventional control theory is actually not intended. However, since the structural properties of the network, which has to be trained are well known, one takes advantage of the approximation ability of neural nets and directly applies control theory methods as learning algorithms. For this purpose, Singhal and Wu [SW89] suggested the use of the *Extended Kalman Filter* algorithm from the *estimation theory* for training of the nets.

5.1 State Model of a Neural Network

The process behaviour is characterized by a *state model* with a *transition* and an *observation* function, cf. sec. 2.1. In case of a neural net, the state model is defined by:

a) State Transition Function

The state transition of the neural network is governed by:

$$\vec{w}(n+1) = \vec{w}(n) = \vec{w}_0; \quad n > 0\,, \tag{33a}$$

where $\vec{w}(n)$ is the *global weight vector* at time step (n).

Throughout training, the state of the network model is defined by the elements of the global weight vector $\vec{w}$. Thus, the *transition function* in eqn. (33a) is the *identity function.* The global weight vector considers all trainable weights in the network. In case that the net converges, the final global weight vector is denoted by $\vec{w}_0$.

b) Observing Function

The output $\vec{d}(n)$ of the model is determined by the *observation function*:

$$\begin{aligned}\vec{d}(n) &= h_n(\vec{w}(n-1), \vec{u}(n)) + \vec{e}(n)\,, &(33b)\\ &= \vec{y}_d(n) + \vec{e}(n)\,, &(33c)\end{aligned}$$

where $\vec{d}(n)$ is the *observed net output*, $\vec{y}_d(n)$ is the *desired net output*, $\vec{u}(n)$ is the *external net input* and $\vec{e}(n)$ is the *observation noise* at time step n. The function $h_n(\cdot)$ is the arbitrary, time-variant, non-linear, continuous observation function.

In eqn. (33c), the actual output of the network $\vec{d}(n)$ is split up into the *desired output* $\vec{y}_d(n)$ and the *observation noise* $\vec{e}(n)$. This representation of the network output introduces the error as *noise* to the model and is one key assumption the application of estimation theory to train the network. The noise is very often assumed to be Gaussian.

5.2 The Global Extended Kalman Filter Algorithm

The major goal of *estimation theory* is to give the best possible prediction $\hat{x}(n)$ of the state vector of an arbitrary process, based on past observations $\vec{y}(s)$, $s \leq n$. This is equivalent with the stochastic problem of calculating the *expected value* of the *state random variable* of the process, under the condition that the past observation values are known, cf. [KK85].

A well known method in estimation theory is the *Kalman Filter (KF).* This algorithm finds the *optimal* set of parameters for *linear dynamic* systems. Extended versions of the Kalman algorithm can be applied to nonlinear systems by linearizing the system around the current estimate of the parameters. Apart from the gradient, the algorithm considers besides the gradient information also the the dependencies among the weights and the estimation error of the weight parameters. Although this method is computationally complex, it yields a speed-up in training time measured in number of pattern set presentation and obtains more acurate solutions, cf. [SW89]. Since the algorithm by Singhal and Wu considers the dependencies of all the weights with each other for the adaption of a single weight in the network, the algorithm is called the *Global Extended Kalman Filter (GEKF).* Other proposals consider only local, i.e. node level, dependencies,[FPDY92, SPD92] and are less complex, but achieve poorer results.

a) Taylor Series Approximation

To linearize the non-linear model, *taylor series approximation* is used. The non-linear function $h_n(\cdot)$ can be expanded around the current estimate of the parameter vector $\hat{w}(n-1)$, cf. [SPD92]. The *observation function* becomes:

$$\vec{d}(n) = h_n(\hat{w}(n-1), \vec{u}(n)) + H^T(n)(\vec{w}_0 - \hat{w}(n-1)) + \rho(n) + e(n) \tag{34}$$

where

$$H(n) = \left. \frac{\partial h_n(\vec{w}, \vec{u}(n))}{\partial \vec{w}} \right|_{\vec{w}=\hat{w}(n-1)} \tag{35}$$

is the *gradient matrix* and $\rho(n)$ is the residual in the Taylor expansion of $h_n(\cdot)$.

b) Linearized State Model

Using the *linearized* observation function, one gets a new state model:

$$\vec{w} = \vec{w}(n-1) = \vec{w}_0 \tag{36a}$$
$$\vec{d}(n) = H^T(n)\vec{w}_0 + \zeta(n) + e(n) \tag{36b}$$

with

$$\zeta(n) = h_n(\hat{w}(n-1), \vec{u}(n)) - H^T(n)\vec{w}(n-1) + \delta(n). \tag{37}$$

c) Global Extended Kalman Algorithm

Estimating the *weight vector* $\hat{w}(n)$ is equivalent with minimizing the *expected value* of the *mean squarred error* between the *actual weight vector* $w(n)$ and the *estimation*, i.e. the problem is to find the minimum of:

$$E[(w(n) - \hat{w}(n))^T S(w(n) - \hat{w}(n))]. \tag{38}$$

The matrix $S(\cdot)$ is a user-defined, arbitrary, positive-definite, symmetric matrix and allows to weight the elements of $w(n) - \hat{w}(n)$ differently.

The *Global Extended Kalman Algorithm* finds the minimum of eqn. (38) by computing the new estimation $\hat{w}(n)$ recursively out of the previous values $\hat{w}(n-1)$. Deriving the complete algorithm is pretty lenghtly, therefore only the Kalman-Equations are provided. A detailed derivation and proof of the Kalman Filter Algorithm can be found in [KK85, Cat89]. The Kalman-Equations are:

$$K(n) = P(n)H(n) \times [(\eta(n)S(n))^{-1} + H^T(n)P(n)H(n)]^{-1}, \tag{39}$$
$$P(n+1) = P(n) - K(n)H^T(n)P(n) + Q(n), \tag{40}$$
$$\hat{w}(n+1) = \hat{w}(n) + K(n)(d(n) - h(\hat{w}(n), \vec{u}(n))). \tag{41}$$

The initial conditions are $P(0) = \delta^{-1} \cdot I\,;\ \delta > 0\,(\approx 10^2)$.

Apart from only considering the past estimation of the weight vector $\hat{w}(n-1)$, the GEKF also accounts the current *covarianz matrix of the estimation error*, $P(n)$. The elements of $P(n)$ describe the dependences of all weights with each other. Since the GEKF aalgorithm relates the weight adaption to the correlation of the weight, it yields a better convergence behaviour than conventional gradient descent methods, cf. sec. 4. For updating the covarianz matrix, the so-called *Kalman-Gain* matrix $K(n)$ is needed. The matrix $S(n)$ defines in conjunction with the scalar $\eta(n)$ the *learning rate.* Finally, $Q(n)$ is a diagonal covarianz matrix that introduces artificial noise in Kalman recursion. The elements of $Q(n)$ are in the range of 10^{-6} to 10^{-2}. Artificial noise prevents the process from getting stuck in local minima, cf. [PF91].

6 Application and Conclusion

Recurrent multi layer perceptrons have shown to be a powerful neural network architecture. They can incorporate temporal behaviour and are able to approximate arbitrary continuous functions. Moreover, since the results of control theory, like process characterization can be transfered to this network type, this architecture suggests to be especially applicable for numerous engineering tasks. Typical applications can be found in mechanics and telecommunication.

a) Identification in Automotive Environment

Feldkamp, Puskorius et. al. investigated in [FPDY92] the identification and control of an *active suspension system.* The system is modeled as a quarter-car, four-state-variable system. The state variables are body momentum, spring deflection, wheel momentum and tire deflection. Feldkamp et. al. assumed a linear model as well as a nonlinear system behaviour and compared the results.

b) Connection Admission Control in ATM Networks

A promising application of a recurrent multi layer perceptron in telecommunications is presented in [NRK94]. The neural network is part of the *connection admission control (CAC)* function, which has to prevent the ATM network from degeneration, which is caused by too many connections. The neural net receives the traffic parameters of a new individual connection request, i.e. the *mean* and the *peak bitrate*, and estimates the multiplexed bitrate of all connections. Based on this estimation and the actual bitrate available in the ATM net, the CAC function decides whether the connection request is acepted or not. From the control theory point, in this application, the neural network can be viewed as an identification model of the multiplexed bitrate.

c) Dynamic Channel Allocation in Mobile Cellular Networks

So far in mobile cellular networks, the channel assignment to cells is performed at the installation of the network and remains static throughout operation. As a result, first of all, the *spectrum efficiency* of the mobile network is very low, and secondly the network can not react on varying teletraffic. The former problem can be resolved by *spatial reuse* of frequencies in the cells. Nevertheless, one needs to consider the *channel interference* and thus only certain *reuse patterns* are allowed. The second problem can be tackled by allocating the channels *dynmically* to the cells, e.g. on demand. A cell with a high offered traffic gets more channels than a cell with low traffic. A *channel allocation function*, which considers both problems, is very complex. Since a combinatorial solution of the function is NP-complete, neural networks have been considered for this task, [CPE91]. However the proposed neural nets perform a static mapping of a current frequency allocation and the new request to a new channel occupancy. They don't consider the *dynamical* varying traffic situation. Now, one can propose to use recurrent multi layer perceptrons to incorporate the temporal behaviour. The network has to be trained by a sequence which consists out of the allocating pattern of the adjacent cells, of the local cell and the call request in the local cell. The neural network then has to produce an efficient (e.g. low blocking probability for new calls) allocation sequence for the cell it controlls.

Acknowledgement: The author would like to thank Daimler Benz Research for the financial support of this study. I also like to give an appreciation to the people of F1/ME for the fruitful discussions during my stay, especially to H.H. Hemberger and W. Stiltz.

References

[Cat89] D. E. Catlin. *Estimation, control, and the discrete Kalman filter.* Springer-Verlag, New York, 1989.

[CBG90] S. Chen, S. Billings, and P. Grant. Non-linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214, 1990.

[CPE91] P. Chan, M. Palaniswami, and D. Everitt. Dynamic channel assignment for cellular mobile radio systems using feedforward neural networks. In *Proceedings of the 1991 IEEE/INNS Int. Joint Conference on Neural Networks (IJCNN), Singapore*, volume II, pages 1242–1247, November 1991.

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[FKK92] D. Franke, K. Krüger, and M. Knoop. *Systemdynamic und Regelerentwurf.* R. Oldenburg Verlag, München, 1992.

[FPDY92] L. Feldkamp, G. Puskorius, L. Davis, Jr., and F. Yuan. Neural control systems trained by dynamic gradient methods for automotive applications. In *IJCNN*, volume 2, pages 798–804, 1992.

[FPT90] B. Fernandez, A. Parlos, and W. Tsai. Nonlinear dynamic system identifikation using artificial neural networks (ANNs). In *Proceedings of the International Joint Conference on Neural Networks*, pages II–133–141, 1990.

[Gee89] H. P. Geering. *Meß- und Regelungstechnik.* Springer-Verlag, Berlin, 1989.

[HKP91] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, Redwood City, Ca., 1991.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[Jor86] M. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Conference of the Cognitive Science S ociety 1986*, pages 531–546. Cognitive Science Society, 1986.

[KK85] H. W. Knobloch and H. Kwakernaak. *Lineare Kontrolltheorie.* Springer-Verlag, Berlin, 1985.

[NP90] K. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[NP91] K. Narendra and K. Parthasarathy. Gradient methods for the optimazation of dynamical systems containing neural networks. *IEEE Transactions on Neural Networks*, 2(2):252–262, March 1991.

[NRK94] T. Necker, T. Renger, and H. Kröner. Bitrate management in ATM systems using recurrent neural networks. In *GLOBECOM'94*, San Franzisko, 27.11.–2.12. 1994.

[NvdS90] H. Nijmeijer and A. van der Schaft. *Nonlinear Dynamical Control Systems.* Springer-Verlag, New York, 1990.

[PCA94] A. Parlos, K. Chong, and A. Atiya. Application of the recurrent multilayer perceptron in modeling complex process dynamics. *IEEE Transactions on Neural Networks*, 5(2):255–266, March 1994.

[PF91] G. Puskorius and L. Feldkamp. Decoupled extended Kalman filter training of feedforward layered networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages I–771–777, Seatlle, 1991.

[PF93] G. Puskorius and L. Feldkamp. Practical considerations for Kalman filter training of recurrent neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages III–1189–1195, 1993.

[PF94] G. Puskorius and L. Feldkamp. Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, 5(2):279–297, March 1994.

[PSY88] D. Psaltis, A. Sideris, and A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, pages 17–21, April 1988.

[RMt86] D. Rumelhart, J. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, Mass. USA, 1986.

[SPD92] S. Shah, F. Palmieri, and M. Datum. Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural networks*, 5:779–787, 1992.

[SW89] S. Singhal and L. Wu. Training multilayer perceptrons with the extended Kalman algorithm. In D. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 133–140. Morgan Kaufman, 1989.

[WZ89] R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[30] Th. Fritsch und W. Mandel. *Communication network routing using neural nets – numerical aspects and alternative approaches.* Juli 1991.

[31] H. Vollmer und K. W. Wagner. *Classes of counting functions and complexity theoretic operators.* August 1991.

[32] R. V. Book, J. H. Lutz und K. W. Wagner. *On complexity classes and algorithmically random languages.* August 1991.

[33] F. Hübner. *Queueing analysis of resource dispatching and scheduling in multi-media systems.* September 1991.

[34] H. Gold und G. Bleckert. *Analysis of a batch service system with two heterogeneous servers.* September 1991.

[35] H. Vollmer und K. W. Wagner. *Complexity of functions versus complexity of sets.* Oktober 1991.

[36] F. Hübner. *Discrete-time analysis of the output process of an ATM multiplexer with periodic input.* November 1991.

[37] P. Tran-Gia und O. Gropp. *Structure and performance of neural nets in broadband system admission control.* November 1991.

[38] G. Buntrock und K. Loryś. *On growing context-sensitive languages.* Januar 1992.

[39] K. W. Wagner. *Alternating machines using partially defined "AND" and "OR".* Januar 1992.

[40] F. Hübner und P. Tran-Gia. *An analysis of multi-service systems with trunk reservation mechanisms.* April 1992.

[41] U. Huckenbeck. *On a generalization of the bellman-ford-algorithm for acyclic graphs.* Mai 1992.

[42] U. Huckenbeck. *Cost-bounded paths in networks of pipes with valves.* Mai 1992.

[43] F. Hübner. *Autocorrelation and power density spectrum of ATM multiplexer output processes.* September 1992.

[44] F. Hübner und M. Ritter. *Multi-service broadband systems with CBR and VBR input traffic.* Oktober 1992.

[45] M. Mittler und P. Tran-Gia. *Performance of a neural net scheduler used in packet switching interconnection networks.* Oktober 1992.

[46] M. Kowaluk und K. W. Wagner. *Vector language: Simple description of hard instances.* Oktober 1992.

[47] B. Menzel und J. Wolff von Gudenberg. ΔKommentierte Syntaxdiagramme für C++. November 1992.

[48] D. Emme. *A kernel for funtion definable classes and its relations to lowness.* November 1992.

[49] S. Öhring. *On dynamic and modular embeddings into hyper de Bruijn networks.* November 1992.

[50] K. Poeck und M. Tins. *An intelligent tutoring system for classification problem solving.* November 1992.

[51] K. Poeck und F. Puppe. *COKE: Efficient solving of complex assignment problems with the propose-and-exchange method.* November 1992.

[52] Th. Fritsch, M. Mittler und P. Tran-Gia. *Artificial neural net applications in telecommunication systems.* Dezember 1992.

[53] H. Vollmer und K. W. Wagner. *The complexity of finding middle elements.* Januar 1993.

[54] O. Gihr, H. Gold und S. Heilmann. *Analysis of machine breakdown models.* Januar 1993.

[55] S. Öhring. *Optimal dynamic embeddings of arbitrary trees in de Bruijn networks.* Februar 1993.

[56] M. Mittler. *Analysis of two finite queues coupled by a triggering scheduler.* März 1993.

[57] J. Albert, F. Duckstein, M. Lautner und B. Menzel. *Message-passing auf transputer-systemen.* März 1993.

[58] Th. Stock und P. Tran-Gia. *Basic concepts and performance of high-speed protocols.* März 1993.

[59] F. Hübner. *Dimensioning of a peak cell rate monitor algorithm using discrete-time analysis.* März 1993.

[60] G. Buntrock und K. Loryś. *The variable membership problem: Succinctness versus complexity.* April 1993.

[61] H. Gold und B. Frötschl. *Performance analysis of a batch service system working with a combined push/pull control.* April 1993.

[62] H. Vollmer. *On different reducibility notions for function classes.* April 1993.

[63] S. Öhring und S. K. Das. *Folded Petersen Cube Networks: New Competitors for the Hyepercubes.* Mai 1993.

[64] S. Öhring und S. K. Das. *Incomplete Hypercubes: Embeddings of Tree-Related Networks.* Mai 1993.

[65] S. Öhring und S. K. Das. *Mapping Dynamic Data and Algorithm Structures on Product Networks.* Mai 1993.

[66] F. Hübner und P. Tran-Gia. *A Discrete-Time Analysis of Cell Spacing in ATM Systems.* Juni 1993.

[67] R. Dittmann und F. Hübner. *Discrete-Time Analysis of a Cyclic Service System with Gated Limited Service.* Juni 1993.

[68] M. Frisch und K. Jucht. ΔPascalli-P. August 1993.

[69] G. Buntrock. *Growing Context-Sensitive Languages and Automata.* September 1993.

[70] S. Öhring und S. K. Das. *Embeddings of Tree-Related Topologies in Hyper Petersen Networks.* Oktober 1993.

[71] S. Öhring und S. K. Das. *Optimal Communication Primitives on the Folded Petersen Networks.* Oktober 1993.

[72] O. Rose und M. R. Frater. *A Comparison of Models for VBR Video Traffic Sources in B-ISDN.* Oktober 1993.

[73] M. Mittler und N. Gerlich. *Reducing the Variance of Sojourn Times in Queueing Networks with Overtaking.* November 1993.

[74] P. Tran-Gia. *Discrete-Time Analysis Technique and Application to Usage Parameter Control Modelling in ATM Systems.* November 1993.

[75] F. Hübner. *Output Process Analysis of the Peak Cell Rate Monitor Algorithm.* Januar 1994.

[76] K. Cronauer. *A Criterion to Separate Complexity Classes by Oracles.* Januar 1994.

[77] M. Ritter. *Analysis of the Generic Cell Rate Algorithm Monitoring ON/OFF-Traffic.* Januar 1994.

[78] K. Poeck, D. Fensel, D. Landes und J. Angele. *Combining KARL and Configurable Role Limiting Methods for Configuring Elevator Systems.* Januar 1994.

[79] O. Rose. *Approximate Analysis of an ATM Multiplexer with MPEG Video Input.* Januar 1994.

[80] A. Schömig. *Using Kanban in a Semiconductor Fabrication Environment — a Simulation Study.* März 1994.

[81] M. Ritter, S. Kornprobst und F. Hübner. *Performance Analysis of Source Policing Architectures in ATM Systems.* April 1994.

[82] U. Hertrampf, H. Vollmer und K. W. Wagner. *On Balanced vs. Unbalanced Computation Trees.* Mai 1994.

[83] M. Mittler und A. Schömig. ΔEntwicklung von „Due–Date“–Warteschlangendisziplinen zur Optimierung von Produktionssystemen. Mai 1994.

[84] U. Hertrampf. *Complexity Classes Defined via k-valued Functions.* Juli 1994.

[85] U. Hertrampf. *Locally Definable Acceptance: Closure Properties, Associativity, Finiteness.* Juli 1994.

[86] O. Rose und M. R. Frater. *Delivery of MPEG Video Services over ATM.* August 1994.

[87] B. Reinhardt. ΔKritik von Symptomerkennung in einem Hypertext-Dokument. August 1994.

[88] U. Rothaug, E. Yanenko und K. Leibnitz. *Artificial Neural Networks Used for Way Optimization in Multi-Head Systems in Application to Electrical Flying Probe Testers.* September 1994.

[89] U. Hertrampf. *Finite Acceptance Type Classes.* Oktober 1994.

[90] U. Hertrampf. *On Simple Closure Properties of #P.* Oktober 1994.

[91] H. Vollmer und K.W. Wagner. *Recursion Theoretic Characterizations of Complexity Classes of Counting Functions.* November 1994.

[92] U. Hinsberger und R. Kolla. *Optimal Technology Mapping for Single Output Cells.* November 1994.

[93] W. Nöth und R. Kolla. *Optimal Synthesis of Fanoutfree Functions.* November 1994.

[94] M. Mittler und R. Müller. *Sojourn Time Distribution of the Asymmetric $M/M/1//N$ – System with LCFS-PR Service.* November 1994.

[95] M. Ritter. *Performance Analysis of the Dual Cell Spacer in ATM Systems.* November 1994.

[96] M. Beaudry. *Recognition of Nonregular Languages by Finite Groupoids.* Dezember 1994.

[97] O. Rose und M. Ritter. *A New Approach for the Dimensioning of Policing Functions for MPEG-Video Sources in ATM-Systems.* Januar 1995.

[98] T. Dabs und J. Schoof *A Graphical User Interface For Genetic Algorithms.* Februar 1995.

[99] M. R. Frater und O. Rose. *Cell Loss Analysis of Broadband Switching Systems Carrying VBR Video.* Februar 1995.

[100] U. Hertrampf, H. Vollmer und K. W. Wagner. *On the Power of Number-Theoretic Operations with Respect to Counting.* Januar 1995.

[101] O. Rose. *Statistical Properties of MPEG Video Traffic and their Impact on Traffic Modeling in ATM Systems.* Februar 1995.

[102] M. Mittler und R. Müller. *Moment Approximation in Product Form Queueing Networks.* Februar 1995.

[103] D. Rooß und K. W. Wagner. *On the Power of Bio-Computers.* Februar 1995.

[104] N. Gerlich und M. Tangemann. *Towards a Channel Allocation Scheme for SDMA-based Mobile Communication Systems.* Februar 1995.

[105] A. Schömig und M. Kahnt. △Vergleich zweier Analysemethoden zur Leistungsbewertung von Kanban Systemen. Februar 1995.

[106] M. Mittler, M. Purm und O. Gihr. *Set Management: Synchronization of Prefabricated Parts before Assembly.* März 1995.

[107] A. Schömig und M. Mittler. *Autocorrelation of Cycle Times in Semiconductor Manufacturing Systems.* März 1995.

[108] A. Schömig und M. Kahnt. *Performance Modelling of Pull Manufacturing Systems with Batch Servers and Assembly-like Structure.* März 1995.

[109] M. Mittler, N. Gerlich und A. Schömig. *Reducing the Variance of Cycle Times in Semiconductor Manufacturing Systems.* April 1995.

[110] A. Schömig und M. Kahnt. *A note on the Application of Marie's Method for Queueing Networks with Batch Servers.* April 1995.

[111] F. Puppe, M. Daniel und G. Seidel. △Qualifizierende Arbeitsgestaltung mit tutoriellen Expertensystemen für technische Diagnoseaufgaben. April 1995.

[112] G. Buntrock, und G. Niemann. *Investigations on Weak Growing Context-Sensitive Grammars.* Mai 1995.

[113] J. García and M. Ritter. *Determination of Traffic Parameters for VPs Carrying Delay-Sensitive Traffic.* Mai 1995.

[114] M. Ritter. *Steady-State Analysis of the Rate-Based Congestion Control Mechanism for ABR Services in ATM Networks.* Mai 1995.

[115] H. Graefe. △Konzepte für ein zuverlässiges Message-Passing-System auf der Basis von UDP. Mai 1995.

[116] A. Schömig und H. Rau. *A Petri Net Approach for the Performance Analysis of Business Processes.* Mai 1995

[117] K. Verbarg *Approximate Center Points in Dense Point Sets.* Mai 1995

[118] K. Tutschku *Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications.* Juni 1995