An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer

S. Ergezinger, *Member, IEEE*, and E. Thomsen

***Abstract*— Multilayer perceptrons are successfully used in an increasing number of nonlinear signal processing applications. The backpropagation learning algorithm, or variations hereof, is the standard method applied to the nonlinear optimization problem of adjusting the weights in the network in order to minimize a given cost function. However, backpropagation as a steepest descent approach is too slow for many applications.**

In this paper a new learning procedure is presented which is based on a linearization of the nonlinear processing elements and the optimization of the multilayer perceptron layer by layer. In order to limit the introduced linearization error a penalty term is added to the cost function.

The new learning algorithm is applied to the problem of nonlinear prediction of chaotic time series. The proposed algorithm yields results in both accuracy and convergence rates which are orders of magnitude superior compared to conventional backpropagation learning.

I. Introduction

THE use of multilayer perceptrons for nonlinear signal processing tasks includes applications like pattern recognition, identification and control of dynamical systems, system modeling and nonlinear prediction of time series [13], [14], [18], to name just a few. The discovery and popularization of the BackPropagation learning rule (BP-learning) has strongly stimulated the research on this type of network. However, the standard BP-learning algorithm described by Rumelhart suffers from the typical handicaps of all steepest descent approaches: Very slow convergence rate and the need for predetermined learning parameters limit the practical use of this algorithm.

Since the publication of Rumelhart [26] in 1986, many improved learning algorithms have been reported in literature. Some use heuristic rules to find optimal learning parameters [1], [31]. Others refine the gradient descent method to accelerate convergence [17]. Further approaches employ different nonlinear optimization methods like conjugate gradients [1], [18], Newton's method or quasi-Newton techniques [12].

All these improvements achieve better convergence rates and for many purposes they perform sufficiently. However, for applications which require high precision outputs, like the prediction of chaotic time series, the known algorithms are often still too slow and inefficient.

Manuscript received August 2, 1993; revised December 26, 1993.

S. Ergezinger was with the University of Hannover, Institut für Allgemeine Nachrichtentechnik, Hannover, Germany. He is now with E-Plus Mobilfunk GmbH, Düsseldorf, Germany.

E. Thomsen was with the University of Hannover, Institut für Allgemeine Nachrichtentechnik, Hannover, Germany. He is now with Mannesmann Mobilfunk GmbH, Düsseldorf, Germany.

IEEE Log Number 9400103.

To reduce the described difficulties we propose a new learning algorithm which does not rely upon the evaluation of local gradients. The new algorithm is based on an optimization of the multilayer perceptron layer by layer. Therefore it will be called Optimization Layer by Layer-learning algorithm (OLL-learning). By optimizing the connection weights of each layer individually the total learning process can be carried out in a much more efficient way. The OLL-learning algorithm reduces the optimization of each layer to a linear problem which can be solved exactly. To limit the unavoidable linearization error a special penalty term is added to the cost function. Layers are optimized alternately in an iterative process. There are no learning parameters which have to be tuned by the user.

In this paper we will concentrate on the problem of nonlinear prediction of chaotic time series.

This problem is well suited to serve as a benchmark on comparing convergence rates and the accuracy of different learning algorithms available for multilayer networks.

In Section II we introduce the problem of nonlinear prediction of chaotic time series and the approach to solve it with a multilayer perceptron. In Section III, we develop our new learning algorithm. Experimental results and a comparison of OLL-learning with different versions of BP-learning and Conjugate Gradient optimization algorithms are presented in Section IV.

II. Neural Networks and Dynamical Systems

Multilayer feedforward networks are shown to learn internal representations allowing them to represent complex nonlinear mappings. Cybenko [6] even proved that MultiLayer Perceptrons (MLP) with only one hidden layer and sigmoidal hidden layer activation functions are capable of approximating any continuous function to within an arbitrary accuracy. Closely related to functional approximation is the application of neural networks for nonlinear signal prediction and forecasting [13], [18].

This problem can be stated as follows:

Suppose we are given a scalar time series $\{u^k\}_{k=1}^N$ with $u^k \in \mathbb{R}$ generated by a (possibly chaotic) dynamical system and we want to predict future outputs of the system under observation.

The state evolution of a discrete dynamical system within its d_S-dimensional state space is defined by a map

$$\tilde{\mathbf{F}} : \mathbb{R}^{d_S} \to \mathbb{R}^{d_S} \qquad \tilde{\mathbf{x}}^{k+1} = \tilde{\mathbf{F}}(\tilde{\mathbf{x}}^k); \quad k \in \mathbb{N} \qquad (1)$$

where $\tilde{\mathbf{x}}^k \in \mathbb{R}^{d_S}$ is the present state of the system, mapped to the next state $\tilde{\mathbf{x}}^{k+1}$ by the vector valued state equations $\tilde{\mathbf{F}}$.

Repeated applications of $\tilde{\mathbf{F}}$ produce a sequence of points $\{\tilde{\mathbf{x}}^k\}_{k=1}^{\infty}$ called an orbit. In order to model the system and to predict future outputs u^k with $k > N$ we have to reconstruct the unknown dynamical equations (1) on the basis of the observed time series $\{u^k\}_{k=1}^{N}$.

From this point of view $\{u^k\}_{k=1}^{N}$ represents a projection of an orbit onto a single coordinate axis of the d_S-dimensional state space.

According to the approach of Packard et al [24] which was put on a firm mathematical basis by Mañé [22] and Takens [30] a reconstruction of the state space, referred to as the d_E-dimensional embedding space, can be achieved by the method of *time delayed coordinates.*

The reconstruction function $\mathbf{F}_r: \mathbb{R} \to \mathbb{R}^{d_E}$ is defined by

$$x^k = \left[u^k, u^{k+\tau}, \cdots, u^{k+(d_E-1)\tau}\right]^T \tag{2}$$

with $\tau \in \mathbb{N}$ for the discrete case.

Assuming that the dynamical evolution of the underlying system takes place on a (strange) attractor of (fractal) dimension d_A, the formal results of Mañé and Takens require

$$d_E \geq 2d_A + 1 \tag{3}$$

as the dimension of the embedding space. A necessary requirement of course is $d_E \geq d_A$. However, choosing d_E according to (3) ensures (2) to be an embedding. Note that the dimension d_S of the original state space is generally unknown and different from the embedding dimension d_E.

The embedded vectors $\{\mathbf{x}^k\}_{k=1}^{N-(d_E-1)\tau}$ with $\mathbf{x}^k \in \mathbb{R}^{d_E}$ represent points of an orbit in the reconstructed state space of the system.

The dynamics in the reconstructed state space are characterized analogous to (1) by defining another map

$$\mathbf{F}: \mathbb{R}^{d_E} \to \mathbb{R}^{d_E} \qquad \mathbf{x}^{k+1} = \mathbf{F}(\mathbf{x}^k). \quad k \in \mathbb{N} \tag{4}$$

The problem is now to determine the nonlinear function $\mathbf{F}$ which gives rise to the given sequence of reconstructed state vectors $\{\mathbf{x}^k\}$ in embedding space. Then it is possible to predict future outputs $u^{N+1}, u^{N+2}, \ldots$ of the observed system simply by iteratively applying $\mathbf{F}$ to map the last known reconstructed state vector $\mathbf{x}^k$ to the future state vectors $\mathbf{x}^{k+1}, \mathbf{x}^{k+2}, \ldots$.

To be precise we only need the last component F_{d_E} of $\mathbf{F}$ which defines the last component $x_{d_E}^{k+1}$ of $\mathbf{x}^{k+1} \in \mathbb{R}^{d_E}$

$$F_{d_E}: \mathbb{R}^{d_E} \to \mathbb{R} \qquad x_{d_E}^{k+1} = F_{d_E}(\mathbf{x}^k). \quad k \in \mathbb{N} \tag{5}$$

This can easily be seen by writing down the last known and the first unknown state vector:

$$\mathbf{x}^{N-(d_E-1)\tau} = \left[u^{N-(d_E-1)\tau}, \cdots, u^N\right]^T \tag{6}$$

$$\mathbf{x}^{N-(d_E-1)\tau+1} = \left[u^{N+1-(d_E-1)\tau}, \cdots, u^{N+1}\right]^T. \tag{7}$$

Since the time series $\{u^k\}$ is known up to $k = N$ the only unknown component in (7) is u^{N+1}. The idea is to use an artificial neural network to learn the desired mapping F_{d_E}: $\mathbb{R}^{d_E} \to \mathbb{R}$ as described above. The trained network then models the dynamical behavior of the observed system and can be used to predict future output on the basis of past measurements.

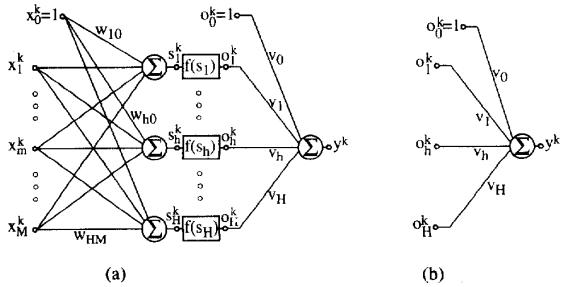


Fig. 1. (a) Multilayer perceptron with one hidden layer and a scalar output. (b) The last layer of the multilayer perceptron (Section III-A).

A number of different network architectures have been applied to the problem of predicting chaotic time series i.e., Lapedes and Farber [18], Farmer and Sidorowich [13], [14] Moody and Darken [23] and Poggio and Girosi [25].

In this paper we will concentrate on the use of multilayer perceptrons. In order to present a straightforward derivation of the new OLL-learning in the next section we only consider MLP's with one hidden layer.

The network is assumed to have M input nodes, one hidden layer with H neurons and a scalar output node. The activation functions are sigmoidal functions of the form

$$f(s) = \frac{1}{1+e^{-s}} \tag{8}$$

for the hidden neurons and linear functions

$$f(s) = s \tag{9}$$

for the output neuron. In order to describe the thresholds of the hidden and the output nodes in the same manner as all other weights the node x_0 with fixed value 1.0 is added to the input layer and the node o_0 with fixed value 1.0 is added to the hidden layer. The resulting network is depicted in Fig. 1(a).

The $(M+1)\cdot(H)$ weight matrix $\mathbf{W} = \{w_{hm}\}$ connects the input and the hidden layer. The $(H+1)$ weight vector $\mathbf{v} = \{\mathbf{v}_h\}$ connects the hidden neurons with the output neuron. The MLP defines a continuous nonlinear mapping $G: \mathbb{R}^M \to \mathbb{R}$ from an input vector $\mathbf{x}^k \in \mathbb{R}^M$ to a scalar output $y^k \in \mathbb{R}$ given by:

$$\begin{aligned} y^k &= \sum_{h=0}^{H} v_h o_h^k = v_0 + \sum_{h=1}^{H} v_h f(s_h^k) \\ &= v_0 + \sum_{h=1}^{H} v_h f\left(\sum_{m=0}^{M} w_{hm} x_m^k\right). \end{aligned} \tag{10}$$

The optimization task is to train the network in such a way that the mapping G performed by the network accurately approximates the mapping F_{d_E} of the dynamical system.

III. Learning with OLL

To characterize the desired mapping a training set is constructed from the series of reconstructed state vectors $\{\mathbf{x}^k\}$. This set consists of K input–output pairs $\{\mathbf{x}^k, d^k\}_{k=1}^{K}$ where $\mathbf{x}^k$ represents the d_E-dimensional input vector to the network

and d^k the desired output value of the network. The remaining state vectors $\{\mathbf{x}^k\}_{k=K+1}^{N-(d_E-1)}$ serve as a test sequence to demonstrate the performance of the trained network when faced with unknown data vectors.

The cost function to be minimized is the mean squared prediction error:

$$E(\mathbf{W},\mathbf{v}) = \frac{1}{K}\sum_{k=1}^{K} E^k \quad \text{with} \quad E^k(\mathbf{W},\mathbf{v}) = \frac{1}{2}(d^k - y^k)^2. \tag{11}$$

For a given network structure and training set (11) defines a nonlinear optimization problem in the network weights $\mathbf{W}$ and $\mathbf{v}$.

If BP-learning is used the weights in one layer are changed independently of each other according to the partial derivative of each weight. The error is propagated back from the output to the input layer. All weights are changed simultaneously. This takes place either after each training pattern, for the on-line version of BP-learning, or after the whole training set is presented to the network, if the batch-version of BP-learning is used [15].

The basic ideas of the new learning algorithm we are proposing are:

- — the weights in each layer are modified dependent on each other but separately from all other layers.
- — in this case the optimization of the output layer is a linear problem
- — the optimization of the weights $\mathbf{W}$ of the hidden layer is reduced to a linear problem by linearization of the sigmoidal activation functions (8).

We call the new learning algorithm OLL. In the following the different steps of the OLL-learning algorithm are described in detail for the network structure depicted in Fig. 1(a).

Note that a generalization of the algorithm to networks with more than one hidden layer and vectoral output is straightforward.

A. Optimization of the Output Layer

As mentioned above the layers are optimized independently of each other. Thus the weights $\mathbf{W}$ of the hidden layer are constants as far as the optimization of the weights $\mathbf{v}$ of the output layer is concerned. Consequently, the outputs $\mathbf{o}^k = (o_0^k, o_1^k, o_2^k, \cdots, o_H^k)^T$ for $k = 1, \cdots, K$ of the hidden neurons are regarded as constant values.

In this setting the output layer is equivalent to an ADALINE (ADAptive LINear Element) or affine combiner as shown in Fig. 1(b).

The output of the affine combiner is simply

$$y^k = \sum_{h=0}^{H} v_h o_h^k = \mathbf{v}^T\mathbf{o}^k. \tag{12}$$

With (11) and (12) the optimization of the output layer can be written as:

$$E(\mathbf{v}^{\text{opt}}) = \min_{\mathbf{v}} E(\mathbf{v}) \quad \text{with}$$

$$E(\mathbf{v}) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{2}(d^k - \mathbf{v}^T\mathbf{o}^k)^2. \tag{13}$$

To derive the optimal weights $\mathbf{v}^{\text{opt}}$ the gradient of the cost function $E(\mathbf{v})$ with respect to $\mathbf{v}$ is calculated and set to zero:

$$\nabla_{\mathbf{v}} E(\mathbf{v}) = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{v}^T\mathbf{o}^k - d^k)\mathbf{o}^k = \mathbf{O} \tag{14}$$

Thus, the set of $(H+1)$ linear equations

$$\mathbf{A}\mathbf{v} = \mathbf{b} \tag{15}$$

has to be solved to find the optimal weight vector $\mathbf{v}^{\text{opt}}$

$$\mathbf{v}^{\text{opt}} = \mathbf{A}^{-1}\mathbf{b} \tag{16}$$

where the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ are given by:

$$\mathbf{A} = \{a_{ij}\}; \quad a_{ij} = \sum_{k=1}^{K} o_i^k o_j^k \quad i,j = 0, \cdots, H \tag{17}$$

$$\mathbf{b} = \{b_i\}; \quad b_i = \sum_{k=1}^{K} d^k o_i^k. \quad i = 0, \cdots, H \tag{18}$$

The square $(H+1)\cdot(H+1)$ matrix $\mathbf{A}$ is symmetric and positive-definite. This fact can be used to solve the matrix inversion in an efficient manner. Note, that $\mathbf{v}^{\text{opt}}$ represents the optimal output layer weights for the current value of $\mathbf{W}$.

B. Linearization of the Neural Network

As shown in the previous section, a set of linear equations has to be solved to find the optimal weights $\mathbf{v}^{\text{opt}}$ for the output layer.

If it is possible to linearize the nonlinear part of the network, i.e., the sigmoidal activation functions, a similar set of linear equations would define the optimal weight matrix $\mathbf{W}^{\text{opt}}$ for the input-to-hidden layer connections, at least as long as the introduced linearization errors are small.

Due to the linearization error and the optimization layer by layer the network optimization can not be carried out in a single step per layer. An iterative procedure which alternately optimizes the output layer and the hidden layer has to be incorporated to compute the weights $\mathbf{v}^*$ and $\mathbf{W}^*$ which minimize the cost function (11).

Assuming the optimization of the input-to-hidden layer connections, see Fig. 1(a), yields a change Δw_{hm}, the updated values are computed according to:

$$w_{\text{new},hm} = w_{\text{old},hm} + \Delta w_{hm}. \qquad m = 0, \cdots, M \quad h = 1, \cdots, H \tag{19}$$

Consequently, the new inputs to the hidden nodes for the k-th training pattern are given by:

$$\begin{aligned} s_{\text{new},h}^k &= \sum_{m=0}^{M} w_{\text{new},hm} x_m^k \\ &= \sum_{m=0}^{M} w_{\text{old},hm} x_m^k + \sum_{m=0}^{M} \Delta w_{hm} x_m^k \\ &= s_{\text{old},h}^k + \Delta s_h^k. \end{aligned} \tag{20}$$

A weight change in the input-to-hidden layer connections Δw_{hm} causes the changes Δs_h^k of the input to the sigmoidal activation functions.

To compute the effects of Δw_{hm} on the new network output y^k_{new}, first the new outputs $o^k_{\text{new},h}$ of the nonlinear activation functions have to be evaluated. A linear dependency between Δw_{hm} and the new network output y^k_{new} is achieved by linearizing the sigmoidal activation functions $f(s)$.

The Taylor series expansion of $f(s)$ about the point $s^k_{\text{new},h}$ is given by:

$$\begin{aligned} f(s^k_{\text{new},h}) &= f(s^k_{\text{old},h} + \Delta s^k_h) \\ &= f(s^k_{\text{old},h}) + \frac{\partial f(s^k_{\text{old},h})}{\partial s^k_{\text{old},h}} \Delta s^k_h + \frac{\partial^2 f(s^k_{\text{old},h})}{2\partial^2 s^k_{\text{old},h}} (\Delta s^k_h)^2 + R \end{aligned} \tag{21}$$

where R represents the higher order terms. Hence, for small values of Δs^k_h the output of the activation functions can be approximated by a first order Taylor series:

$$o^k_{\text{new},h} = \begin{cases} f(s^k_{\text{old},h}) + \frac{\partial f(s^k_{\text{old},h})}{\partial s^k_{\text{old},h}} \Delta s^k_h & \text{for } h = 1, \cdots, H \\ o^k_{\text{old},h} = 1 & \text{for } h = 0. \end{cases} \tag{22}$$

Using this approximation we get the new network output

$$\begin{aligned} y^k_{\text{new}} &= \sum_{h=0}^{H} v_h o^k_{\text{new},h} \\ &\approx \sum_{h=0}^{H} v_h o^k_{\text{old},h} + \sum_{h=1}^{H} f'(s^k_{\text{old},h}) v_h \Delta s^k_h \\ &= y^k_{\text{old}} + \Delta y^k. \end{aligned} \tag{23}$$

For the optimization of the hidden layer weight matrix $\mathbf{W}$ only the changes

$$\Delta s^k_h = \sum_{m=0}^{M} \Delta w_{hm} x^k_m \quad \text{for } h = 1, \cdots, H \quad \text{and } k = 1, \cdots, K. \tag{24}$$

and Δy^k caused by $\Delta \mathbf{W}$ have to be considered.

Defining

$$v^k_{\text{lin},h} := f'(s^k_{\text{old},h}) \cdot v_h \quad \text{for } h = 1, \cdots, H \text{ and } k = 1, \cdots, K \tag{25}$$

the change in the network output can be written as

$$\Delta y^k = \sum_{h=1}^{H} v^k_{\text{lin},h} \Delta s^k_h \quad \text{for} \quad k = 1, \cdots, K. \tag{26}$$

The resulting linearized network structure is depicted in Fig. 2.

Note that the weights $v^k_{\text{lin},h}$ in the output layer now depend on the training pattern $\{\mathbf{x}^k, d^k\}$ just being processed.

C. Optimization of the Hidden Layer

The purpose of optimizing the weight matrix $\mathbf{W}$ of the hidden layer is to further reduce the mean squared error

$$E = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{2} (d^k - y^k_{\text{old}})^2 = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{2} (e^k_{\text{old}})^2 \tag{27}$$

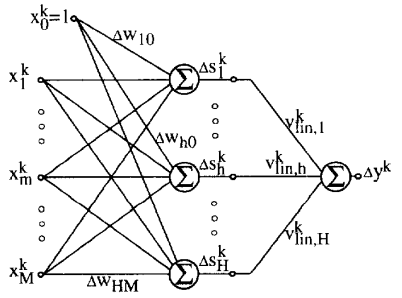


Fig. 2. The linearized network structure for the optimization of the hidden layer connections.

measured for the original network of Fig. 1(a) with optimized output layer weights $\mathbf{v}^{\text{opt}}$ as described in Section III-A. To achieve this reduction the linearized network of Fig. 2 is trained to predict the errors e^k_{old} of the original network.

Thus, we use the modified training set $\{\mathbf{x}^k, e^k_{\text{old}}\}_{k=1}^{K}$ and define a new cost function

$$E_{\text{lin}} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{2} (e^k_{\text{old}} - \Delta y^k)^2 \tag{28}$$

for the optimization of $\Delta \mathbf{W}$ in the linearized network.

As stated above the introduced linearization error has to be kept quite small. Then the network of Fig. 2 is a suitable approximation for the behavior of the MLP (Fig. 1(a)) as far as weight changes $\Delta \mathbf{W}$ are concerned.

In the linearized network the activation functions of the hidden neurons were approximated by a first order Taylor series expansion. Thus, the approximation errors consist of the second and higher order terms of the Taylor series.

For small Δs^k_h the quadratic term of the Taylor series expansion

$$\epsilon^k_h = \frac{1}{2} f''(s^k_{\text{old},h}) (\Delta s^k_h)^2 \tag{29}$$

is obviously a good estimate for the error caused by computing the linearized output values of the hidden neurons using (22).

The conventional way of formulating the constrained minimization problem

$$\begin{aligned} &E_{\text{lin}}(\Delta \mathbf{W}^{\text{opt}}) = \min_{\Delta \mathbf{W}} E_{\text{lin}}(\Delta \mathbf{W}) \\ &\text{subject to} \quad \epsilon^k_h(\Delta \mathbf{W}) \le \epsilon_{\max}; \quad h = 1, \cdots, H; \quad k = 1, \cdots, K \end{aligned} \tag{30}$$

is not feasible here because of the huge number of $H \cdot K$ constraints. Therefore it is more advisable to modify the new cost function (28) by adding a penalty term to limit the linearization error. By multiplying ϵ^k_h with the output weight v_h of the MLP we can account for the influence of the linearization error on the network output y^k. The introduced penalty term estimates the quality of the linear approximation by averaging the absolute values $|\epsilon^k_h v_h|$ over all hidden nodes and all training patterns.

$$E_{\text{pen}} = \frac{1}{K\,H} \sum_{k=1}^{K} \sum_{h=1}^{H} |\epsilon^k_h v_h|. \tag{31}$$

The modified cost function for the optimization of the hidden layer is given by

$$E_{\text{hid}} = E_{\text{lin}} + \mu E_{\text{pen}} \tag{32}$$

with E_{lin} according to (28).

The weighting factor μ determines the influence of the penalty term E_{pen} against the linear cost function E_{lin}.

A correct choice of μ is important for a fast convergence of the entire optimization procedure. If μ is too small, large weight changes $\Delta\mathbf{W}$ are possible since the linearization error will not be limited sufficiently by E_{pen}. Although the hidden layer cost function E_{hid} will be minimized, the prediction error of the original MLP will in most cases increase since the linearization is not valid. On the other hand, if μ is too large only very small weight changes $\Delta\mathbf{W}$ are allowed because of the strong influence of the penalty term E_{pen}. In this case the linearization is valid and the cost function of the MLP is decreased in each iteration step, but convergence is very slow.

The best value for μ lies between these two extremes and cannot be derived from the optimization itself. Therefore, a heuristic procedure for the choice and the adaptation of μ is given at the end of this section. For the following calculations μ is treated as a constant factor.

The minimization of the hidden layer cost function E_{hid} is similar to the minimization of (13) presented in Section III-A. The computation of the partial derivatives of E_{hid} is carried out separately for E_{lin} and E_{pen}.

First, the derivatives of E_{lin} with respect to the weight changes Δw_{hm} are determined.

$$\frac{\partial E_{\text{lin}}}{\partial \Delta w_{hm}} = \frac{1}{K}\sum_{k=1}^{K}\frac{\partial E_{\text{lin}}^k}{\partial \Delta w_{hm}} = \frac{1}{K}\sum_{k=1}^{K}(\Delta y^k - e_{\text{old}}^k)\cdot\frac{\partial \Delta y^k}{\partial \Delta w_{hm}}. \tag{33}$$

Using (24) and (26) to replace Δy^k results in:

$$\frac{\partial E_{\text{lin}}}{\partial \Delta w_{hm}} = \frac{1}{K}\sum_{k=1}^{K}\left[\left(\sum_{j=1}^{H} v_{\text{lin}_j}^k \sum_{i=0}^{M} x_i^k \Delta w_{ji}\right) - e_{\text{old}}^k\right] \times x_m^k v_{\text{lin}_h}^k. \tag{34}$$

Changing the order of summations to

$$\frac{\partial E_{\text{lin}}}{\partial \Delta w_{hm}} = \sum_{i=0}^{M}\sum_{j=1}^{H}\Delta w_{ji}\cdot\frac{1}{K}\sum_{k=1}^{K} v_{\text{lin}_h}^k v_{\text{lin}_j}^k x_m^k x_i^k - \frac{1}{K}\sum_{k=1}^{K} e_{\text{old}}^k x_m^k v_{\text{lin}_h}^k \tag{35}$$

the following averages over the training set can be defined

$$a_{hm,ji} := \sum_{k=1}^{K} v_{\text{lin}_h}^k x_m^k \cdot v_{\text{lin}_j}^k x_i^k \quad ;$$

$$\tilde{b}_{hm} := \sum_{k=1}^{K} e_{\text{old}}^k \cdot v_{\text{lin}_h}^k \cdot x_m^k \tag{36}$$

Now the derivatives of E_{lin} can be written as

$$\frac{\partial E_{\text{lin}}}{\partial \Delta w_{hm}} = \frac{1}{K}\left[\sum_{i=0}^{M}\sum_{j=1}^{H}\Delta w_{ji}\cdot a_{hm,ji} - \tilde{b}_{hm}\right] \quad \text{for } m = 0,\cdots,M;\quad h = 1,\cdots,H. \tag{37}$$

The partial derivatives of the penalty term E_{pen} (31) are calculated using (29) to replace the linearization errors ϵ_h^k.

$$\frac{\partial E_{\text{pen}}}{\partial \Delta w_{hm}} = \frac{1}{K\,H}\sum_{k=1}^{K}\sum_{j=1}^{H}|v_j\cdot f''(s_{\text{old}_j}^k)|\cdot\Delta s_j^k\cdot\frac{\partial \Delta s_j^k}{\partial \Delta w_{hm}} \tag{38}$$

Substituting Δs_j^k using (24) and calculating the derivative yields

$$\frac{\partial E_{\text{pen}}}{\partial \Delta w_{hm}} = \frac{1}{K\,H}\sum_{i=0}^{M}\Delta w_{hi}|v_h|\cdot\sum_{k=1}^{K}|f''(s_{\text{old}_h}^k)|\cdot x_i^k x_m^k. \tag{39}$$

Again, defining the average over the training set

$$c_{hm,i} := |v_h|\cdot\sum_{k=1}^{K}|f''(s_{\text{old}_h}^k)|\cdot x_m^k x_i^k \tag{40}$$

simplifies the partial derivatives of E_{pen} to

$$\frac{\partial E_{\text{pen}}}{\partial \Delta w_{hm}} = \frac{1}{K\,H}\sum_{i=0}^{M}\Delta w_{hi}\cdot c_{hm,i} \quad \text{for}\quad h = 1,\cdots,H;\quad m = 0,\cdots,M. \tag{41}$$

To find the optimal weight changes $\Delta\mathbf{W}^{\text{opt}}$ all partial derivatives of the hidden layer cost function E_{hid} (32) with (37) and (41) are set to zero.

$$\frac{\partial E_{\text{hid}}}{\partial \Delta w_{hm}} = 0 = \sum_{i=0}^{M}\sum_{j=1}^{H} a_{hm,ji}\Delta w_{ji} + \frac{\mu}{H}\sum_{i=0}^{M} c_{hmi}\Delta w_{hi} - \tilde{b}_{hm}. \tag{42}$$

Using the abbreviation

$$\tilde{a}_{hm\ ji} := \begin{cases} a_{hm\ ji} & \text{for}\quad j \neq h \\ a_{hm\ ji} + \frac{\mu}{H}\,c_{hm\ i} & \text{for}\quad j = h \end{cases} \tag{43}$$

we get a set of $(M+1)\cdot H$ linear equations which defines the optimal weight changes Δw_{hm} for the hidden layer connections.

If the matrix of weight changes $\Delta\mathbf{W}^{\text{opt}}$ is interpreted as a $(M+1)\cdot H$ dimensional vector $\Delta\mathbf{W}^{\text{opt}} = (\Delta w_{10}^{\text{opt}},\cdots, \Delta w_{1M}^{\text{opt}},\cdots,\Delta w_{H0}^{\text{opt}},\cdots,\Delta w_{HM}^{\text{opt}})^T$ (44) can be written in matrix notation as found at the bottom of the next page:

$$\tilde{\mathbf{A}}\cdot\Delta\mathbf{W}^{\text{opt}} = \tilde{\mathbf{b}}. \tag{45}$$

Solving this set of linear equations yields the optimal weight changes:

$$\Delta\mathbf{W}^{\text{opt}} = \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{b}}. \tag{46}$$

The matrix $\tilde{\mathbf{A}}$ is symmetric and positive-definite. Its dimension equals the number of connections in the hidden layer.

Especially if larger networks are considered it is important to solve (45) for ΔW^{opt} using a computationally efficient algorithm. The experimental results presented in Section IV were performed with an implementation of the proposed OLL-learning algorithm which incorporated a Cholesky decomposition to solve (45) and (15) directly. This method is quite efficient and turned out to be numerically stable. Additionally, if the symmetries of (36) are utilized, only a small number of matrix elements $\tilde{a}_{hmji}$ have to be calculated.

It should again be noted that the optimal ΔW^{opt} of (46) is only valid for the linearized network and the modified cost function (32). To find the optimal weights W^* and v^* which minimize the cost function (11) of the original MLP an iterative procedure is used which alternately optimizes the output layer and the hidden layer.

The whole OLL-learning procedure can be stated as follows:
Optimization Layer by Layer - OLL-learning algorithm

1) **Initialization**:
   - — choose initial values for the MLP connections $\mathbf{W} = \mathbf{W}^{\text{start}}$; $\mathbf{v} = \mathbf{v}^{\text{start}}$
   - — choose an initial value for the weighting factor in (32) $\mu = \mu^{\text{start}}$
   - — choose the maximum number of iterations $\text{IC}_{\max}$
   - — choose the desired prediction error E^{des} of the MLP
   - — set the iteration counter IC = 0
2) **Optimization of the output layer weights (see Section III-A)**
   - — increment the iteration counter IC:=IC+1
   - — compute the optimal output layer weights $\mathbf{v}^{\text{opt}}$ according to
   - — (16) for the present network described by $\mathbf{W}$ and $\mathbf{v}$
   - — update the output layer weights $\mathbf{v} = \mathbf{v}^{\text{opt}}$
   - — compute the prediction error $E = E(\mathbf{W}, \mathbf{v})$ according to (11) for the MLP with weights $\mathbf{W}$ and $\mathbf{v}$
3) **Optimization of the hidden layer weights (see Section III-C)**
   a. — compute the optimal weight changes $\Delta\mathbf{W}^{\text{opt}}$ by minimizing (32) with the weight factor μ
      — compute the prediction error E^{test} according to (11) for the MLP with weights $\mathbf{W}^{\text{test}} = \mathbf{W} + \Delta\mathbf{W}^{\text{opt}}$ and $\mathbf{v}$
   b. IF $E^{\text{test}} < E$ THEN
      - — update the hidden layer connections $\mathbf{W} := \mathbf{W} + \Delta\mathbf{W}^{\text{opt}}$
      - — set the prediction error of the MLP to $E = E^{\text{test}}$
      - — decrease the influence of the penalty term by decreasing μ $\mu := \mu \cdot \beta$ with $0 < \beta < 1$
      - — continue with step 4
      - — ELSE
      - — increase the influence of the penalty term by increasing μ $\mu := \mu \cdot \gamma$ with $\gamma > 1$
      - — continue with step 3.b
4) **Test for completion**
   IF ($\text{IC} < \text{IC}_{\max}$) AND ($E > E^{\text{des}}$) THEN
   - — continue with step 2

   ELSE
   - — save the final network parameters $W^* = W$ and $v^* = v$
   - — end of OLL-learning

In all experiments presented in Section IV the parameters controlling the adaptation of μ were fixed to the following values:
$\mu^{\text{start}} = 10^{-4}$, $\gamma = 1.2$ and $\beta = 0.9$

IV. Experimental Results

This section demonstrates the performance of the proposed OLL-learning algorithm when applied to the optimization of MLP's to predict chaotic time series. We present results obtained for modeling the dynamics of the Lorenz model and the Mackey–Glass equations. All computations were performed using double-precision floating-point numbers conforming to IEEE 754.

Firstly, as an example for a dynamical system with a three-dimensional state space ($d_S = 3$) we integrated the Lorenz equations [19]

$$\begin{aligned}\frac{dx_1(t)}{dt} &= \sigma \cdot (x_2(t) - x_1(t))\\ \frac{dx_2(t)}{dt} &= r \cdot x_1(t) - x_2(t) - x_1(t)x_3(t)\\ \frac{dx_3(t)}{dt} &= x_1(t)x_2(t) - b \cdot x_3(t)\end{aligned} \tag{47}$$

using a fourth-order Runge–Kutta method with a step size of 0.01 seconds and an initial state vector $\mathbf{x}^0 = (0.0, 0.01, 0.0)$.

$$\begin{aligned}
\tilde{a}_{10\ 10}\Delta w_{10} + \tilde{a}_{10\ 11}\Delta w_{11} + \cdots + \tilde{a}_{10\ 20}\Delta w_{20} + \cdots + \tilde{a}_{10\ HM}\Delta w_{HM} &= \tilde{b}_{10}\\
\tilde{a}_{11\ 10}\Delta w_{10} + \tilde{a}_{11\ 11}\Delta w_{11} + \cdots + \tilde{a}_{11\ 20}\Delta w_{20} + \cdots + \tilde{a}_{11\ HM}\Delta w_{HM} &= \tilde{b}_{11}\\
\vdots \qquad\qquad & \qquad \vdots\\
\tilde{a}_{20\ 10}\Delta w_{10} + \tilde{a}_{20\ 11}\Delta w_{11} + \cdots + \tilde{a}_{20\ 20}\Delta w_{20} + \cdots + \tilde{a}_{20\ HM}\Delta w_{HM} &= \tilde{b}_{20}\\
\vdots \qquad\qquad & \qquad \vdots\\
\tilde{a}_{HM\ 10}\Delta w_{10} + \tilde{a}_{HM\ 11}\Delta w_{11} + \cdots + \tilde{a}_{HM\ 20}\Delta w_{20} + \cdots + \tilde{a}_{HM\ HM}\Delta w_{HM} &= \tilde{b}_{HM}
\end{aligned} \tag{44}$$

Choosing the parameters to $\sigma = 16$, $r = 45,92$ and $b = 4$ yields chaotic behavior. The initial 3000 state vectors were not used to avoid transients.

The time series presented to the MLP consisted of the x_1-components of the following 101 000 state vectors. An estimate of the attractor dimension $d_A(lor)$ is given by the correlation dimension which is $d_C(lor) = 2.06$ for the Lorenz attractor with the here chosen parameters [11]. According to Takens' theorem this implies an embedding dimension $d_E(lor)$ in the range from 3 to 7. For more details on the dimension of chaotic attractors see Eckmann and Ruelle [8] and Parker and Chua [5]. An algorithm for computing the correlation dimension from experimental data is due to Grassberger and Procaccia [11].

For the second system with chaotic behavior we selected the Mackey–Glass delay-differential equation [21]:

$$\frac{dx(t)}{dt} = \frac{a \cdot x(t - \Delta T)}{1 + x(t - \Delta T)^{10}} - b \cdot x(t). \tag{48}$$

This system was first investigated by Mackey and Glass [21] and is now frequently used as a benchmark for nonlinear predictors [7], [18], [23]. If the parameters $a = 0.2$ and $b = 0.1$ are chosen, the asymptotic behavior of the system changes with increasing values of ΔT from equilibrium and periodic solutions to chaotic behavior. Due to the time delay ΔT the state space of this system is infinite dimensional. However, the stationary dynamical evolution takes place on a low dimensional attractor.

We used an initial constant value of $x(t) = 0.8$ for $t \leq 0$ and integrated the Mackey–Glass equation using a fourth-order Runge–Kutta method with a step size of 1.0. The first 3000 values were again not used.

To compare our results with other publications we computed a time series containing 101 000 samples for time delay ΔT= 30 referred to as mg_{30}-time series. According to [18] the fractal dimension of the mg_{30}-attractor is $d_A(mg_{30}) = 3.5$. Takens' theorem indicates an embedding dimension in the range $d_E(mg_{30}) = 4 \cdots 9$.

A. A Simple Optimization Problem

For a first comparison between OLL- and BP-learning we used the MLP depicted in Fig. 3 to predict the chaotic Lorenz time series one time step into the future. The applied network had two inputs, one hidden, and one output node. We call this a 2:1:1 network. Consequently, the embedding dimension d_E was two. The time delay was chosen to $\tau = 1$ sample. The training set consisted of 1000 input–output pairs $\{\mathbf{x}^k, d^k\} = \{(u^k, u^{k+1})^T, u^{k+2}\}$ constructed of the first 1002 samples of the Lorenz time series. An embedding dimension of 2 is definitely too small for reconstructing the Lorenz attractor, but this trial run only served as a graphical comparison.

Optimizing the network depicted in Fig. 3 represents a 5-dimensional nonlinear minimization problem. To find the optimal network weights we used OLL-learning and a variation of BP-learning, known as Bold-Driver-method (BD-method) [1], [31] in combination with a momentum term (momentum constant $\alpha = 0.9$) [15], [26]. The concept of the BD-algorithm

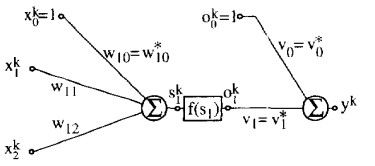


Fig. 3. The 2:1:1 MLP-network used to visualize the convergence properties of different learning algorithms.

is to raise or lower the learning rate η depending on whether the cost function is decreased or increased.

The quality of the predictors was measured in terms of the Normalized Root-Mean Squared Error (NRMSE) in dB:

$$\frac{\text{NRMSE}}{[\text{dB}]} = 20 \log_{10} \frac{\langle \| d^k - y^k \|^2 \rangle^{0.5}}{\langle \| d^k - \langle d^k \rangle \|^2 \rangle^{0.5}}. \tag{49}$$

Both algorithms were run until optimum weights $\mathbf{w}^*$ were found, which for this low dimensional optimization problem were the same. The achieved NRMSE was -41.83 dB for the training set. Due to the high precision we demanded the number of iterations was quite high for this simple problem. The optimum weight vectors $\mathbf{w}^*$ were computed within 15 000 iterations of the OLL-learning algorithm and within 1 380 000 iterations of the BD-algorithm. This result indicates a speedup of 92 as far as the number of iterations is concerned. Since OLL-learning is computationally more complex we also compared the CPU-time for both runs. For the chosen 2:1:1 network the BD-algorithm is 2.1 times faster per iteration than OLL-learning. Thus, the effective speedup is almost 44 for this low dimensional optimization problem. A comparison of the convergence speed for larger networks is given in Sections IV-B and IV-C.

To visualize the convergence behavior of different optimization techniques the network parameters w_{10}, v_0 and v_1 were fixed to their optimal values. This leaves the weights (w_{11}, w_{12}) of the hidden layer to be trained by the algorithms. Choosing the initial weights $(0.0, 0.0)$ it took OLL-learning three iterations to adjust the network weights to their optimal values (w_{11}^*, w_{12}^*). Fig. 4(a) shows a contour plot of the normalized error surface in conjunction with the weights after each iteration of OLL-learning.

For comparison we employed the batch-mode version of standard BP-learning with as well as without ($\alpha = 0.0$) momentum term for fixed learning rates in the range $10^{-2} \leq \eta \leq 10^2$.

The results for various values of η and α are summarized in the first four columns of Table I. The results obtained with BP-learning for values of $\eta \leq 1$ are not included because convergence is even slower. For values of $\eta \geq 20$ the batch-mode version of BP-learning did not converge at all.

In addition the last four columns of Table I contain the number of iterations necessary to compute the optimum weights (w_{11}^*, w_{12}^*) with BD- and OLL-learning for different initial weight combinations. Fig. 4(b) shows the weight combinations after each iteration of BD-learning with momentum constant $\alpha = 0.99$ and initial weights $(0.0, 0.0)$ within a contour plot of the weight space. The ravine in weight space is reached at ($w_{11} = 0.9$, $w_{12} = 0.9$) after only 80 iterations. Then the

TABLE I

	w_{11}= 0.0 w_{12}= 0.0				w_{11}= 1.0 w_{12}= 0.75	w_{11}=-5.5 w_{12}= 2.0	w_{11}=-5.5 w_{12}= 0.0
	Backpropagation-Algorithm						
	$\eta = 1.0$	$\eta = 2.0$	$\eta = 5.0$	$\eta = 10.0$	*Bold-Driver*		
$\alpha = 0.0$	194862	97429	38969	19483	14846	14955	13634	9869
$\alpha = 0.1$	175374	87685	35071	17533	11918	12005	10917	7918
$\alpha = 0.3$	136398	68195	27273	13633	9007	9070	8243	5996
$\alpha = 0.5$	97420	48703	19473	9730	6542	6549	5981	4339
$\alpha = 0.7$	58434	29204	11666	5820	4879	4794	4406	3265
$\alpha = 0.9$	19397	9654	3806	1855	2435	2461	2285	1719
$\alpha = 0.99$	12433	5751	2125	1381	1250	1249	1172	896
	OLL-Learning-Algorithm							
	3					5	6	3

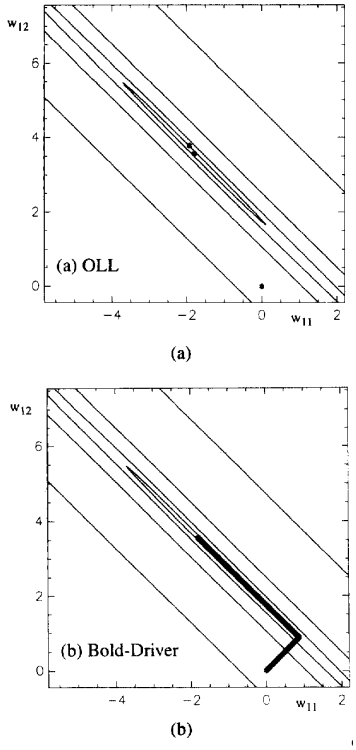


Fig. 4. Contour plots of NRMSE over the hidden layer weights in conjunction with the weight combinations after each iteration. The contour levels are −20, −10, 0, and 10 dB, respectively. The initial coordinates are ($w_{11} = 0.0, w_{12} = 0.0$). (a): OLL-learning finds the optimal weight combination after three iterations. (b): the BD-method with $\alpha = 0.99$ needs 1250 iterations to find the exact minimum.

gradient descent optimization method spends more than 1100 iterations to find its way through the ravine in weight space to the optimal weight combination. The reason for this behavior is the well known *Zig-Zagging* effect [20] caused by the fact that the gradient is shallow in the direction of the minimum but steep in the other directions.

The convergence performance of BD- and OLL-learning for different initial weights is in principle the same for all initial coordinates. OLL-learning finds the exact minimum after three to six iterations. Even for the best choice of the initial coordinates and the momentum constant the BD-method needs 896 iterations to compute (w_{11}^*, w_{12}^*).

Due to the special form of the error surface a value of α very close to 1.0 yields the best results. This doesn't hold for larger networks with more complex error surfaces. In literature a range of α between 0.6 and 0.9 is suggested for larger optimization problems.

This simple example already reveals some advantages of the proposed learning algorithm:

- — the convergence rate is extremely high
- — there are no user adjustable optimization parameters besides the embedding parameters and the network structure.
- — the optimization is not based upon special assumptions concerning the form of the error surface.
- — the influence of the randomly chosen initial network weights on the optimization process is not very strong.

B. Prediction of the Chaotic Lorenz Time Series

As stated above, reconstructing the Lorenz attractor from a scalar time series implies an embedding dimension in the range of 3 to 7. Therefore, training a MLP with only two input-nodes will not yield the best possible predictor for the chaotic Lorenz signal.

We performed a number of experiments varying the embedding parameters $2 \leq d_E \leq 20$ and $1 \leq \tau \leq 8$ as well as the number of hidden nodes $H \in \{8, 16, 32, 100, 200\}$ and the number of training patterns $K \in \{500, 1000, 2000\}$ to find the optimal predictor for the Lorenz time series. Since OLL-learning shows a very high convergence rate even for moderately large networks with at least several hundred weights we performed only 1000 iterations to train the networks. Additionally we studied the generalization abilities of the trained networks when confronted with unknown data by using the remaining 100 000 samples of the Lorenz time series to form a test sequence for the predictors.

The best prediction results for the Lorenz times series were achieved with a 5:8:1 network and an embedding time delay

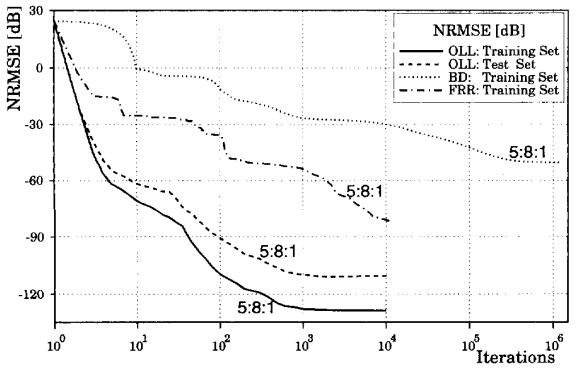


Fig. 5. Learning curves showing the NRMSE in dB against the number of iterations for training a 5:8:1 network to predict the chaotic Lorenz time series.

of $\tau = 1$ samples. The training sequence contained 500 input–output pairs. To investigate the effects of overlearning we performed 10 000 iterations with OLL-learning and computed the NRMSE for the test sequence after each iteration. The achieved NRMSE was -128 dB during training and -110 dB for the unknown 100,000 test patterns. Two curves of Fig. 5 monitor in a log-log plot the NRMSE values for the training- and the test-sequence over the number of iterations for OLL-learning.

We found that standard BP-learning with fixed η had in general a smaller convergence rate than the BD-method. Since there were no significant differences between on-line- and batch-versions the results obtained using the match-mode BD-method with momentum constant $\alpha = 0.8$ are presented in Fig. 5 for a comparison to OLL-learning.

Additionally we investigated a variety of Conjugate-Gradient (CG) and Quasi-Newton (QN) optimization methods [1], [2], [7], [12], [17], [18]. All CG- and QN-methods performed one up to three orders of magnitude better than BD-learning as far as the number of iterations is concerned. However, due to additional line-searches needed by the CG- and QN-methods the computational complexity is 3 to 10 times higher compared to BD-learning. For most experiments the CG-method of Fletcher–Reeves with Restart (FRR) [9], [10], [20] proved to be superior over Polak–Ribière [9], [10], [20], Limited-Quasi-Newton [2], [20], Scaled-Conjugate-Gradients [28] and the CG-method of Shanno [28]. Hence, besides the results of BD- and OLL-learning only the curve obtained using FRR-learning is included in Fig. 5.

Comparing the learning curves of BD-, FRR- and OLL-learning is difficult because the BD-method becomes extremely inefficient after several thousand iterations. We had to use a logarithmic scale for the number of iterations to monitor the results of all three algorithms within one diagram. Because of limited CPU-time we were not able to achieve the same results with the BD- and the FRR-methods which we achieved with OLL-learning.

OLL-learning already saturates after 1000 iterations. A minimum NRMSE for the test sequence is achieved after 4000 iterations. However, the improvement from iteration 1000 to iteration 4000 is only marginal. Due to overlearning [15] further training of the network weights reduces the NRMSE for the **training set** but slowly increases the prediction error for the **test set.**

Even after 1 000 000 iterations the BD-algorithm was about 80 dB worse than OLL-learning after 1000 iterations! After 10 000 iteration FRR-learning achieves the same NRMSE OLL-learning reaches after 27 iterations. Taking into account the CPU time factors OLL/FRR/BD=7.5/5/1 the speed up of OLL-learning over BD-learning is several orders of magnitude and over FRR-learning 2 up to 3 orders of magnitude.

The investigated 5:8:1 network with a total number of 57 weights is still a small MLP. The behavior of OLL-learning for optimizing MLP's with up to 2001 weights is demonstrated in the next subsection.

C. Prediction of the Chaotic Mackey–Glass Time Series

This problem is frequently used in literature to demonstrate and to compare the performance of learning algorithms and network architectures [7], [13], [14], [18], [23], [27].

We used different network topologies and embedding parameters from those reported in Lapedes and Farber [18] or [27]. However, the formulated prediction problem is the same as the one treated in [18], [7], [27]. As in Section IV-B we varied the embedding parameters, the number of hidden nodes and the number of training patterns to find the optimal network to predict the mg_{30} time series generated by the Mackey–Glass equation (48) six time steps into the future.

For this prediction problem we found the embedding parameters $d_E = 8$ and $\tau = 4$ and a MLP with eight hidden nodes to yield the best results. We achieved a NRMSE of -48.9 dB for the training set of 1000 input–output patterns. This 8:8:1 network has a total number of 81 weights compared to the 6:10:10:1 network with 191 weights used by Lapedes and Farber [18]. They achieved a NRMSE value of 0.03 or -30.5 dB, but didn't report the number of iterations needed by the employed CG-optimization to train the network weights. The same prediction problem was also investigated by Sanger [27], who obtained NRMSE of 0.025 or -32 dB with a tree-structured adaptive network. Note that the result of -48.9 dB or 0.0036 is almost an order of magnitude better than those reported by Lapes and Farber [18] or Sanger [27]. The learning curves for the 8:8:1-MLP monitoring the NRMSE values for the **training set** after each iteration of OLL-learning, the FRR-CG-optimization algorithm as well as the BD-method are shown as part of Fig. 6. The comparison of the tree optimization methods yields similar results to those obtained for predicting the Lorenz time series. The computational complexities measured in used CPU-time per iteration scale as follows OLL/FRR/BD=7.7/3.7/1. Taking also the learning curves of Fig. 6 for the 8:8:1 network into account the speed-up of OLL- over FRR-learning is 25 measured after 10^5 iterations and 420 for OLL- compared to BD-learning after 10^6 iterations.

We already mentioned that OLL-learning possesses a high convergence rate even for MLP's with several hundred weights. To demonstrate this property and to show that increasing the number of hidden nodes does not necessarily

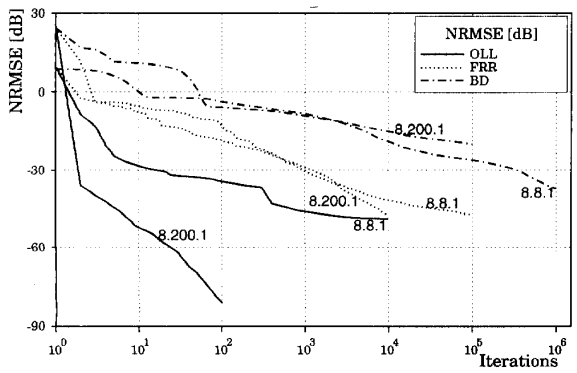


Fig. 6. Learning curves showing the NRMSE for 1000 training patterns over the number of iterations obtained during the optimization of a 8:8:1and a 8:200:1 network. The networks were trained with the OLL-, FRR-CG-, and BD-learning algorithms to predict the mg_{30}-time series six time steps into the future.

induce a significantly decreased prediction error for the test sequence, we increased the number of hidden nodes but left the embedding parameters d_E and τ constant. Again, we compared OLL-, FRR-, and BD-learning. The corresponding learning curves for the 8:200:1 networks are also depicted in Fig. 6.

The convergence rate of OLL- and FRR-learning for the 8:200:1 network is even higher compared to the 8:8:1 network as far as the number of iterations is concerned. In contrary, the convergence rate of BD-learning decreases significantly for the optimization of the larger network. Again, due to limited CPU-time we were not able to achieve the same results with FRR- and BD-learning which we achieved with OLL-learning. The CPU-time factors are about OLL/FRR/BD=145/11/1 for the 8:200:1 network. Taking this higher complexity of OLL-learning into account, the speedup is about 700 compared to BD-learning and 300 compared to FRR-CG-learning.

Fig. 6 shows the NRMSE values computed for the 1,000 training patterns. The corresponding NRMSE values achieved with OLL-learning for the **test set** were -47.4 dB for the 8:8:1 network with 81 weights but only -30.5 dB for the larger 8:200:1 network with 2001 weights. These prediction results indicate a strong overlearning for the larger network. Note that with the chosen embedding parameters 1028 samples are needed to construct the training set. Hence, in the case of the 8:200:1 network there are roughly twice as many weights to be optimized than there are data points in the training set.

We observed that as soon as the order of the optimization problem is close to or exceeds the amount of training data, OLL-learning will result in a network which almost reproduces the training set. In this case the generalization ability of the MLP is very poor.

Consequently, to avoid overlearning, the training sequences should contain al least 5 to 10 times more data points than there are parameters in the MLP.

D. Relation to Other Work

An approach, somehow similar to the concept of OLL-learning was published recently by Biegler-König and Bärmann [3], [4]. Their proposed algorithms BV1 and BV2 (see [4]) also optimize the network weights layer by layer but employ a different coupling between successive layers in order to update the weights.

OLL-learning minimizes the NRMSE directly. The network weights within one layer are optimized according to the internal node values *before* the corresponding sigmoidal activation functions of that layer. The OLL-learning algorithm as presented at the end of Section III guarantees a non-increasing NRMSE.

Instead of linearizing portions of the MLP-network Biegler-König and Bärmann use the inverse activation functions to compute target node values and derive a set of linear equations, which after some computations lead to an update of the network weights. This however results in output values of the hidden layer nodes which are not limited to the range of the employed activation functions. Therefore, the calculated weights have to be transformed to ensure target hidden layer node values in the given range of the activation functions.

In order to compare OLL-learning and the algorithms introduced by Biegler-König and Bärmann with BP-learning the same problem as described in [4] was investigated.

The MLP-network was trained to approximate the nonlinear mapping of eight-dimensional input vectors $\mathbf{x}^k$ to three-dimensional output vectors $\mathbf{d}^k$ given by

$$\mathbf{f}: \mathbb{R}^8 \to \mathbb{R}^3; \quad \mathbf{d}^k = \mathbf{f}(\mathbf{x}^k) \quad \text{with} \quad \mathbf{x}^k \in \mathbb{R}^8; \mathbf{d}^k \in \mathbb{R}^3 \tag{50}$$

with

$$\begin{aligned} d_1 &= \frac{x_1 \cdot x_2 + x_3 \cdot x_4 + x_5 \cdot x_6 + x_7 \cdot x_8}{4} \\ d_2 &= \frac{x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8}{8} \\ d_3 &= \frac{1}{2}\sqrt{4 - x_1 \cdot x_2 + x_3 \cdot x_4 + x_5 \cdot x_6 + x_7 \cdot x_8}. \end{aligned} \tag{51}$$

The learning was based on a training set containing 50 sample input–output vectors $\{\mathbf{x}^k, \mathbf{d}^k\}_{k=1}^{50}$. The components of the input vectors were random numbers equally distributed over the interval [0, 1]. In contrary to the introduced NRMSE (49) Biegler-König and Bärmann used the average absolute error

$$E_{\text{abs}}(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K} E_{\text{abs}}^k$$

with

$$E_{\text{abs}}^k(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} |d_n^k - y_n^k| \tag{52}$$

computed for the test set to measure the quality of different learning algorithms.

In Table II we quote the results for a 8:8:3:3- and a 8:8:8:3-MLP obtained with algorithms BV1, BV2 [4], and BP-learning which were presented by Biegler-König and Bärmann in [4].

For a comparison of these results to OLL-learning a generalized OLL-learning algorithms for MLP's with more than one hidden layer and vectoral output was implemented. As indicated earlier this generalization based on the same basic ideas of OLL-learning presented in Section III is straightforward.

TABLE II

	8:8:3:3-MLP				8:8:8:3-MLP				8:14:3-MLP
It. Nr.	BV1	BV2	BP	OLL	BV1	BV2	BP	OLL	OLL
0	0.1948	0.1948	0.1948	0.920	0.2231	0.2231	0.2231	0.861	1.32
1	0.0171	0.0232	0.0695	0.0313	0.0167	0.0204	0.0738	0.0188	0.0140
3	-	-	-	0.00862	-	-	-	0.00503	0.00451
10	0.0164	0.0172	0.0682	0.00311	0.0163	0.0232	0.0704	0.00169	0.000679
50	-	0.0085	0.0241	0.00130	-	0.0237	0.0241	0.00079	0.000203
500	-	-	0.0161	0.00105	-	0.0087	0.0165	0.00034	0.000103
2000	-	-	0.0124	0.00088	-	-	0.0073	0.00020	0.000058
3000	-	-	0.0084	0.00087	-	-	-	0.00018	0.000052

The results achieved with OLL-learning for both networks as well as a 8:14:3-MLP are additionally included in Table II.

In the case of the 8:8:8:3 MLP OLL-learning achieves after two iterations a value of E_{abs} which BV2 reaches after 500 and BP after almost 2000 iterations

According to [4] and our experiments the computational complexity of the employed learning algorithms scales as OLL/BV1/BV2/BP $= 8 \cdots 9/2/3 \cdots 4/1$ for the investigated MLP-networks.

Taking this into account OLL-learning is still about two orders of magnitude faster than BV1-, BV2-, and BP-learning.

Additionally it should be noted that the 8:14:3-MLP has the same number of network weights (171 weights) as the 8:8:8:3-MLP but yields much better approximation results. The optimum network structure heavily depends on the underlying problem. A MLP with more than one hidden layer is not always superior to a MLP with only one hidden layer.

V. Conclusion

We have presented a new learning algorithm for multilayer perceptrons which optimizes the network weights in an iterative process layer by layer. The proposed OLL-learning algorithm introduces a linearization of the hidden node sigmoidal activation functions for the optimization of the hidden layer weights. In order to take the introduced linearization error into account we added a special penalty term to the cost function of the linearized network. In contrary to BP-, BD-, and CG-learning there are no user adjustable optimization parameters for OLL-learning.

To evaluate the proposed learning algorithm, we trained multilayer perceptrons to predict the chaotic Lorenz- and Mackey–Glass time series. In all experiments presented in Section 4 the OLL-learning algorithm was orders of magnitude faster than conventional backpropagation, the Bold Driver method, Conjugate Gradient methods and to both algorithms proposed by Biegler-König and Bärmann in [3], [4].

The speed-up was even higher for larger networks and for an increased demand in accuracy. Most of the problems with a high demand in accuracy could not be solved satisfactorily with other learning algorithms we investigated.

The presented results indicate that for the prediction of chaotic time series data with an underlying low dimensional attractor multilayer perceptrons with only one hidden layer and a small number of hidden nodes perform sufficiently.

We are aware of the fact that the proposed OLL-learning algorithm involves non local computations. However, we believe, that its high convergence rate and the absence of user adjustable parameters (except the network structure) makes this algorithm suitable for off-line training of multilayer perceptrons.

Since the OLL-learning algorithm is not limited to prediction tasks, we expect it to have high performance in many other applications.

In this paper we described the OLL-learning algorithm for MLP-networks with only one hidden layer and a scalar output. The generalization to networks with an arbitrary number of hidden layers and a vectoral output based on the basic ideas of the presented OLL-learning is straight forward. Recent results as well as the approximation of the nonlinear mapping presented in Section IV-D demonstrate that OLL-learning is well suited for MLP-networks with general structure.

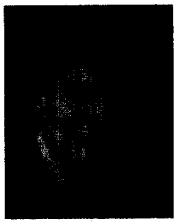
Acknowledgment

Discussions with Mrs. S. Smarsli and Mr. U. Arz on optimization methods for multilayer perceptrons are highly appreciated. The authors would like to thank the reviewers for valuable comments and suggestions and for pointing out the relations to the algorithms proposed in [3], [4] by Biegler-König and Bärmann.

References

[1] R. Battiti, "Accelerating Backpropagation learning, two optimization methods," *Complex Syst.*, vol. 3, pp. 331–342, 1989.
[2] R. Battiti, "First and second-order methods for learning, between steepest descent and Newtons's method," submitted to *IEEE Trans. Neural Net.*
[3] F. Bärmann, F. Biegler-König, "On a class of efficient learning algorithms for neural networks," *Neural Net.*, vol. 5, no. 1, 1992.
[4] F. Biegler-König and F. Bärmann, "On a class of efficient learning algorithms for multi-layered neural networks," in *Artificial Neural Networks*, vol. 2, I. Aleksander and J. Taylor ed. Amsterdam, The Netherlands: Elesvier Science, 1992.
[5] T. S. Parker and L. O. Chua, *Practical Numerical Algorithms for Chaotic Systems*. New York: Springer Verlag, 1989.
[6] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, vol. 2, pp. 303–314, 1989.
[7] S. D. Day and M. R. Davenport, "Continuous-time temporal backpropagation with adaptable time delays," preprint submitted to, *IEEE Trans. Neural Net.*, Aug. 1991

[8] J.-P. Eckmann and D. Ruelle, "Ergodic theory of chaos and strange attractors," *Rev. Mod. Phys.*, vol. 57, pt. I, no. 3, July 1985.
[9] R. Fletcher, "Practical methods of optimization," in *Unconstrained Optimization*. Chichester, U.K.: Wiley, 1980.
[10] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. London, U.K.: Academic Press, 1981.
[11] P. Grassberger and I. Procaccia, "Measuring the strangeness of strange attractors," *Physicia D*, vol. 9, pp. 189–208, 1983.
[12] S. E. Fahlmann, "Faster learning variations on back-propagation, an empirical study," in *Proc. 1988 Connectionists Models Summer School*, pp. 38–51, 1989.
[13] J. D. Farmer and J. J. Sidorowich, "Predicting chaotic time series," *Phys. Rev. Lett.*, vol. 59, no. 8, pp. 845–848, 1987.
[14] J. D. Farmer and J. J. Sidorowich, "Predicting chaotic dynamics," in *Dynamic Patterns in Complex Systems*. World Scientific, 1988.
[15] R. Hecht-Nielsen, *Neurocomputing*. Reading, MA: Addison-Wesley, 1990.
[16] M. Hénon, "A two-dimensional mapping with a strange attractor," *Commun. Math. Phys.*, vol. 50, p. 69, 1976.
[17] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Net.*, vol. 1, pp. 295–307, 1988.
[18] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks, prediction and system modeling," Tech. Rep. LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.
[19] E. N. Lorenz, "Deterministic nonperiodic flow," *J. Atmos. Sci.*, vol. 20, p. 130, 1963.
[20] D. G. Luenberger, *Linear and Nonlinear Programming*.
[21] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, 1977.
[22] R. Mañé, "On the dimension of the compact invariant sets of certain nonlinear maps," in *Dynamical Systems and Turbulence*, D. A. Rand and L.-S. Yo ung, ed. Berlin, West Germany: Springer Verlag, 1981), pp. 230–242.
[23] J. Moody and C. Darken, "Learning with localized receptive fields," in *Proc. Connectionist Models Summer School*, 1988.
[24] N. H. Packard, J. P. Crutchfield, J. D. Farmer ,and R. S. Shaw, "Geometry from a time series," *Phys. Rev. Lett.*, vol. 45, pp. 712–716, 1980.
[25] T. Poggio and F. Girosi, "Networks for Approximation and Learning," *Proc. IEEE*, vol. 78, no. 9, Sept. 1990.
[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in*Parallel Distributed Processing, Exploration in the Microstructure of Cognition*, vol. 1, D. E. Rumelhart and J. L. McClelland, eds. Cambridge, MA: MIT Press, 1986.
[27] T. D. Sanger, "A tree-structured adaptive network for function approximation in high-dimensional spaces," *IEEE Trans. Neural Net.*, vol. 2, Mär. 1991.
[28] D. F. Shanno, "Conjugate Gradient methods with inexact searches," *Math. Oper. Res.*, Band 3, no. 3, S. 244–256, Aug. 1978.
[29] M. Stinchcombe and H. White, "Universal approximation using feed-forward networks with non-sigmoid hidden layer activation functions," in *Int. Joint. Conf. Neural Net.*, June 1989.
[30] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence—Lecture Notes in Math.* 898. Springer Verlag, 1981, pp. 366–381.
[31] T. P. Vogel, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the convergence of the back-propagation method," *Bio. Cybernetics*, vol. 59, pp. 257–263, 1988.



Siegfried Ergezinger (S'89–M'92) received the Diploma in electrical engineering from the University of Hannover, Germany in 1987. The presented new learning algorithm is part of his ongoing work for the Ph. D. degree in electrical engineering.

His research interests include the modeling and coding of speech signals and the application of artificial neural networks to modeling and prediction of time signals. Since April 1993, he has been with the DCS1800 network operator E-Plus Mobilfunk GmbH, Düsseldorf, Germany.

Erk Thomsen received the Diploma in electrical engineering from the University of Hannover, Germany, in 1991. The new learning algorithm presented in this paper was developed during the work on his *Diplomarbeit* (Master's thesis).

He is now working in the mobile communication industry.