Improving Generalization Capabilities of Dynamic Neural Networks

Miroslaw Galicki
galicki@imsid.uni-jena.de
Institute of Medical Statistics, Computer Sciences and Documentation, Friedrich Schiller University, Jena, Germany, and Department of Management, University of Zielona Góra, Zielona Góra, Podgórna 50, Poland

Lutz Leistritz
i6lelu@imsid.uni-jena.de
Institute of Medical Statistics, Computer Sciences and Documentation, Friedrich Schiller University Jena, Germany

Ernst Bernhard Zwick
ernst.zwick@kfunigraz.ac.at
Department of Pediatric Orthopedics, Karl-Franzens-University, Graz, Austria

Herbert Witte
iew@imsid.uni-jena.de
Institute of Medical Statistics, Computer Sciences and Documentation, Friedrich Schiller University, Jena, Germany

This work addresses the problem of improving the generalization capabilities of continuous recurrent neural networks. The learning task is transformed into an optimal control framework in which the weights and the initial network state are treated as unknown controls. A new learning algorithm based on a variational formulation of Pontryagin’s maximum principle is proposed. Under reasonable assumptions, its convergence is discussed. Numerical examples are given that demonstrate an essential improvement of generalization capabilities after the learning process of a dynamic network.

1 Introduction

Artificial neural networks are now commonly used in many practical tasks, such as the classification, identification and control, and fault diagnostic, of dynamic processes. Due to the nature of these processes, it seems more reasonable to apply recurrent networks, which are, in fact, (nonlinear) dynamic systems. A general problem for a neural network in all of these tasks is to find an underlying mapping that correspondingly relates their input-output...
spaces based on a given (usually finite) set called a learning set of input-output pairs. As a rule, the learning pairs present some functions of time (trajectories). It is well known that the problem of finding such a mapping is ill posed (Tikhonov & Arsenin, 1977; Morozov, 1984) because its solution is not unique. Consequently, some criterion reflecting a network performance, which in turn influences the realization quality of the above tasks, should be introduced in order to obtain a unique solution. The capability of a neural network to generalize seems to be of utmost important from both a theoretical and practical point of view. The generalization property is strongly related to the bias and variance—components of the generalization error (Geman, Bienenstock, & Doursat, 1992). In the case of neural networks, these components depend in general on the training algorithm used that finds an optimal solution with respect to the weights, a network structure (an optimal balance between bias and variance), and the power of the learning set.

In this context, several approaches may be distinguished, although they refer for the most part to feedforward neural networks being static systems. An exhaustive overview of the algorithms modifying a network structure during the training of the feedforward networks may be found in the work of Doering, Galicki, & Witte (1997). In the case of recurrent networks, the recurrent cascade correlation method has been proposed by Fahlman (1991) to tackle the problem of network parameter reduction. As was proved in Giles, Chen, and Sun (1995), it is unable to learn some practical tasks. Recently, several training algorithms for dynamic networks with fixed structures have been proposed that find constant optimal weights using, for example a Lyapunov-based stability theory (Fabri & Kadirkamanathan, 1996), an error function of integral type (Pearlmutter, 1995; Cohen, Saad, & Marom, 1997), the expectation-maximization algorithm (Ma & Ji, 1998) or a concept from the theory of stochastic learning automata (Sundareshan & Condarcure, 1998; see Galicki, Leistritz, & Witte, 1999, for a broader review of the learning algorithms). Nevertheless, these procedures influence only variations of the bias. Another approach to the improvement of generalization performance of a neural network is provided by the Vapnik-Chervonenkis (VC) dimension (Vapnik & Chervonenkis, 1971). Roughly speaking, the VC dimension determines conditions under which the bias decreases faster than the variance increases (Abu-Mostafa, 1989). Unfortunately, the resulting worst-case estimation of the number of training pairs required to train a neural network is very large (Holden & Niranjan, 1995). As recurrent networks are Turing machines (Kiliom & Siegelmann, 1996), the corresponding VC dimension may take arbitrarily large values for arbitrarily long input signals (DasGupta & Sontag, 1996; Hammer, 1997). An alternative means of controlling the generalization capability is through the use of regularization techniques that are based on the addition of a penalty term, called a regularizer to the error function. The penalty term influences variations of the network variance. To the best of our knowledge, they have been applied...
for the most part to static (feedforward) neural networks. In this context, three approaches may be distinguished.

The first is based on using a quadratic weight decay regularizer (Hinton, 1987; Czernichow, 1997; Taniguchi & Tresp, 1997; Hansen & Rasmussen, 1994). However, empirical evidence indicates that the regularizer of this form can lead to improvement in the generalization of feedforward networks. Utilizing the concept of small perturbations of the training data, Wu and Moody (1996) and Leung, Tsoi, and Chan (2001) have proposed an improved version of a weight decay regularizer. Moreover, this type of regularizer was analyzed for discrete recurrent networks in Wu and Moody (1996).

In the next approach, involving network information criterion (NIC) (Murata, Yoshizawa, & Amari, 1994; Billings & Zheng, 1995), which is a generalization of Akaike's information criterion (AIC) into neural networks, the penalty term is proportional to the number of weights. As variations of the error function might be much greater than the added penalty term, the NIC may result in a poor generalization capability of a feedforward network.

The third approach, extensively studied by Simard, Victorri, Le Cun, and Denker (1991), Bishop (1991, 1995b), and Girosi, Jones, and Poggio (1995) is based on adding a regularizer to the error function that measures the discrepancy between the actual and desired directional derivative of some selected transformations in Simard et al. (1991) or penalizes the curvature of a feedforward network mapping (Bishop, 1991, 1995b; Girosi, Jones, & Poggio, 1995). In our opinion, this approach seems to be very efficient since it directly influences variations of the variance (this is because directional derivative and curvature-driven methods yield network mappings with lower network mapping variance). Nevertheless, a suitable choice of penalty parameter that controls the bias-variance trade-off is very difficult in practice. As seen from the cited literature, very little has been written with reference to the improvement of generalization capabilities for continuous dynamic neural networks, although a solution to this problem has a crucial effect on the performance of multiple practical tasks.

The purpose of this work is to propose a step toward the construction of learning algorithms satisfying this requirement. The generalized dynamic neural networks (GDNNs), first introduced and theoretically analyzed by Leistritz, Galicki, and Witte (1998) and Galicki, Leistritz, and Witte (1999), seem to be suitable in such a case. They are characterized by time-dependent weights. Introducing time-dependent weights results in an overparameterization of GDNNs. However, this property allows us to define simultaneously a larger number of tasks to be performed, thus reducing or (limiting) network parameter redundancy. More formally, our aim is to construct a learning algorithm by producing a small global error between output trajectories and the desired (target) ones, which, in addition, results in better generalization capabilities of GDNNs. As is known from the theory of static
neural networks, the minimization of their structures leads to a decrease
of the variance. However, it has been numerically shown by Galicki et al.
(1999) that a GDNN of a theoretically minimal size has worse generalization
capabilities than a fully connected one. On the other hand, a weight decay
term indirectly influences the network mapping variance. Consequently,
our approach is based on using either the sensitivity or the curvature of
a GDNN’s mapping as the regularizers added to standard error function.
They directly influence variations of the neural network mapping and hence
its variance, too. As opposed to static networks, the dynamic (time-varying)
curvature or sensitivity is not directly given. An adjoint system of ordinary
differential equations (ODEs) is introduced to determine the network sensi-
tivity or curvature. In order to fully specify the equations, initial conditions
must be found during the training of the dynamic network. By an ana-
logy to the static regularization theory, we both extend and generalize it
to the dynamic case using the adjoint system of ODEs. Nevertheless, the
determination of a penalty parameter may still be difficult in such a case.
However, GDNNs offer another possibility for tackling this problem. As is
known, they possess a very useful property. Galicki et al. (1999) have shown
that it is possible to find the network weights that produce an arbitrarily
small value of the error functional. This property makes it possible to for-
mulate our task as the minimization of the curvature or sensitivity norm
of a GDNN’s mapping subject to a given accuracy of learning. We immedi-
ately notice that the above optimization problem does not involve a penalty
parameter whose determination is difficult in practice. Thus, the learning
tasks defined above may be treated as a problem of dynamic programming
with state constraints in which the weights and initial network state are
treated as unknown controls. In this case, it is very difficult to use Pon-
tryagin’s maximum principle (in its classical form; Pontryagin, Boltyansky,
Gamkrelidze, & Mischenko, 1961) because it presents only the positive form
for control. In fact, the strong variation algorithms based on the maximum
principle (Srochko, 1986) are not oriented to the optimal control problems
involving state inequality constraints. Sakawa and Shindo (1980) have de-
digned an algorithm that handles state variable inequality constraints by
using a prediction technique. Controls are computed by minimizing the
penalized Hamiltonian. Strend and Balden (1990) have shown the ineffec-
tiveness of this algorithm in the case of bang-bang control problems. The
convergence proofs of many implementable algorithms are based on gener-
ating a sequence of controls. However, the optimal control problem with
state inequality constraints may not have a limit control (in a class of mea-
surable controls), although the optimal trajectories for such a problem may
exist (Filippov, 1962). It is natural to attempt to employ other techniques in
order to solve the learning tasks defined above. The variational formulation
of Pontryagin’s maximum principle is used here for this purpose, which
makes it possible to handle state constraints efficiently. The technique of
convergence proof of the proposed algorithm is also discussed. It is differ-
ent from those presented above and is based on searching for limit network trajectories whose existence is theoretically ensured.

The article is organized as follows. Section 2 presents the learning task in terms of the optimal control problem. Section 3 describes how to employ the variational formulation of Pontryagin’s maximum principle to determine optimal weights. Some numerical examples of classification and time-series prediction problems, confirming theoretical results, are presented in section 4. Finally, concluding remarks are drawn in section 5.

2 Problem Formulation

A neural network of the Hopfield type is considered whose state evolves according to the following differential and algebraic equations

\[ \dot{y} = f(y, w, u) \]
\[ z = g(y). \quad (2.1) \]

where \( \mathbb{R}^N \ni y = (y_1, \ldots, y_N)^T \) is a state vector of the network; \( N \) denotes the number of all neurons; \( w = (w_{(i-1)N+j} : 1 \leq i, j \leq N) \) represents the weight connection out of neuron \( i \) to neuron \( j \); \( u = u(t) = (u_1, \ldots, u_N)^T \) stands for the vector of external inputs to the network; \( f = (f_1, \ldots, f_N)^T \); \( f_i(y, w, u) = -y_i + h(\sum_{j=1}^{N} w_{(i-1)N+j}y_j) + u_i; \ i = 1 : N; \ t \in [0, T] \); \( T \) is a fixed time of network evolution; \( y(0) = A = (A_1, \ldots, A_N)^T \) denotes an initial network state; \( h(\cdot) \) is strictly increasing neural activation function; \( g : \mathbb{R}^N \rightarrow \mathbb{R}^L; g(y) \) stands for a given mapping that represents outputs of the network; and \( N \geq L \) is the number of neurons producing the outputs. The mapping \( g \) usually takes on the following form (states of neurons \( 1, \ldots, L \) present outputs of the network):

\[ g(y) = (y_1, \ldots, y_L)^T. \quad (2.2) \]

In order to simplify further considerations, time constants of neurons as well as their synaptic delays will be omitted from now on. The natural prerequisite during the recognition of multiple classes is that the initial state of the network (given a priori or determined during the learning) should be the same for all the external inputs. Nevertheless, the specification (e.g., by the user) of a poor initial state may considerably slow the convergence rate of the learning process or even lead to convergence difficulties (see Galicki et al., 2000, for learning with an unknown initial state). Hence, the learning task of multiple continuous trajectories is now to modify the weight vector \( w \) and the initial state \( y(0) \) so that the output trajectories \( z_p(t) \) corresponding to external inputs \( u_p(t) = (u_{1,p}(t), \ldots, u_{N,p}(t))^T \) follow differentiable desired (target) trajectories \( d_p(t) \in \mathbb{R}^L \), where \( p = 1 : P; 1 \leq P \) denotes a given number of training pairs.
As is known in many practical applications of dynamic neural networks, the available external inputs in the learning set are often noisy. If the error functional equals zero for the learning data, the capability of the network to generalize, that is, to produce an acceptable output when a novel external input is presented, will often be poor. This fact is a consequence of rapid variations in the network mapping that generally are needed to decrease the bias (i.e., to interpolate the noisy data). In order to improve the generalization capability of a dynamic neural network, it seems necessary to control the bias-variance trade-off, or, in other words, to approximate the learning set by the network mapping rather than to interpolate it. This requirement implies that in general, the error functional for the learning data equals no more zero. Nevertheless, approximation prevents large variations of the network variance. One constructive way to limit these variations is to add a penalty term to the error functional whose role is to produce large values for the rapid variations in the network mapping that are characteristic for a poor generalization. Techniques that exploit such types of constraints in learning problems of static neural networks (e.g., perceptrons, radial basis functions) are commonly termed regularization (Girosi et al., 1995; Bishop 1995a). This is frequently used in a number of other fields for tackling ill-posed problems (Tikhonov & Arsenin, 1977).

To the best of our knowledge, using regularization with network curvature smoothing for training the dynamic neural networks has not been addressed so far in the literature, although it is important from both a theoretical and practical point of view. By analogy to the static regularization theory, the following performance index may be introduced in the dynamic case,

$$J = \frac{1}{2} \int_0^T \left( \sum_{p=1}^P \| z_p - d_p \|^2 + \lambda \left\| \frac{\partial^2 z_p}{\partial u^2_p} \right\|^2 \right) dt,$$

(2.3)

where the first term in the brackets stands for the instantaneous error function and the second is a scaled square norm of the network curvature; \( \frac{\partial^2 z_p}{\partial u^2_p} \) is the standard Euclidean norm; and \( \lambda \) is the regularization parameter that controls the compromise between the degree of smoothness of the network mapping and its closeness to the data. Therefore, \( \lambda \) is directly related to the degree of generalization that is enforced. Minimizing the term \( \left\| \frac{\partial^2 z_p(t)}{\partial u^2_p(t)} \right\|^2 \) in equation 2.3 forces the derivative with respect to external input \( u_p(t) \) of the network mapping (its sensitivity) to be locally invariant with respect to small perturbations of \( u_p(t) \) (for a fixed time instant \( t \)). Moreover, small initial network sensitivity value (e.g., equal to zero, as is assumed in all the computer simulations) additionally results in approximately local invariance of the network output for small
Improving Generalization Capabilities of Dynamic Neural Networks

variations of $u_p(t)$ at $t$. Furthermore, using a curvature-driven term instead of a sensitivity one makes it possible to approximate better the learning set by the network mapping. The term $\frac{\partial^2 z_p}{\partial u_p^2}$ in equation 2.3 may be computed based on equation 2.2. It equals

$$\frac{\partial^2 z_p}{\partial u_p^2} = \frac{\partial^2 g}{\partial u_p \partial y_p} \frac{\partial y_p}{\partial u_p} + \frac{\partial g}{\partial y_p} \frac{\partial^2 y_p}{\partial u_p^2},$$

where

$$\frac{\partial y_p}{\partial u_p} = \left[ \frac{\partial y_{p,i}}{\partial u_{p,j}} \right]_{1 \leq i,j \leq N}, \quad \frac{\partial^2 y_p}{\partial u_p^2} = \left[ \frac{\partial^2 y_{p,i}}{\partial u_{p,k} \partial u_{p,j}} \right]_{1 \leq i,j,k \leq N}.$$

On account of the fact that $\frac{\partial^2 g}{\partial u_p \partial y_p} = 0$, we obtain

$$\frac{\partial^2 z_p}{\partial u_p^2} = \frac{\partial g}{\partial y_p} \frac{\partial^2 y_p}{\partial u_p^2},$$

and hence the performance index, equation 2.3, takes the form

$$J = \frac{1}{2} \int_0^T \left( \sum_{p=1}^P \| z_p - d_p \|^2 + \lambda \left\| \frac{\partial g}{\partial y_p} \frac{\partial^2 y_p}{\partial u_p^2} \right\|^2 \right) \, dt. \quad (2.4)$$

Consequently, the computation of $J$ from equation 2.4 requires the knowledge of the second derivative $\frac{\partial^2 y_p}{\partial u_p^2}$ as a function of time. It may be found as follows. First, let us differentiate equation 2.1 with respect to $u_p$ (assuming that the corresponding partial derivatives are continuous). The result is

$$\frac{d}{dt} \left( \frac{\partial y_p}{\partial u_p} \right) = \frac{df}{du_p}, \quad (2.5)$$

where

$$\frac{df}{du_p} = \left[ \frac{df_i}{du_{p,j}} \right] = \frac{\partial f}{\partial y_p} \frac{\partial y_p}{\partial u_p} + \frac{\partial f}{\partial u_p}.$$

Differentiating the last equation by $u_p$ for the second time, we obtain the adjoint system of ODEs,

$$\frac{d}{dt} \left( \frac{\partial^2 y_p}{\partial u_p^2} \right) = \frac{d^2 f}{du_p^2}. \quad (2.6)$$
where

\[
\frac{d^2 f}{du_p^2} = \left[ \frac{d^2 f_i}{du_{p,i}^2} \right] = \left( \frac{\partial y_p}{\partial u_p} \right)^T \frac{\partial^2 f}{\partial y_p^2} \frac{\partial y_p}{\partial u_p} + \frac{\partial^2 f}{\partial y_p \partial u_p} \frac{\partial y_p}{\partial u_p} + \frac{\partial^2 f}{\partial y_p^2} \frac{\partial y_p}{\partial u_p} + \frac{\partial^2 f}{\partial u_p^2} \frac{\partial y_p}{\partial u_p}.
\]

Let us note that the solution of equation 2.6 is possible only when the term \(\frac{\partial y_p}{\partial u_p}\) is known as a function of time. It may be found based on equation 2.5.

In order to fully specify differential equations 2.5 and 2.6, initial values for derivatives \(\frac{\partial y_p}{\partial u_p}\) and \(\frac{\partial^2 y_p}{\partial u_p^2}\), that is,

\[
\frac{\partial y_p}{\partial u_p}(0) = B
\]

\[
\frac{\partial^2 y_p}{\partial u_p^2}(0) = C,
\]

(2.7)

where \(B = [B_{ij}]_{1 \leq i,j \leq N}\); \(C = [C_{ijk}]_{1 \leq i,j,k \leq N}\) denote some constant matrices, should be given. However, they are not known in practice. Therefore, an additional task arises, which consists of determining initial values 2.7 when learning dynamic network 2.1. A potential disadvantage of applying criterion 2.4 is that it requires knowledge of the regularization parameter \(\lambda\). This may be cumbersome from a practical point of view since this parameter is problem dependent. Therefore, an alternative performance index may be proposed in the form

\[
J = \frac{1}{2} \int_0^T \sum_{p=1}^P \left\| \frac{\partial g}{\partial y_p} \frac{\partial^2 y_p}{\partial y_p \partial u_p} \right\|^2 dt,
\]

(2.8)

which reflects only the generalization capability of the network. Thus, the learning task may now be formulated more compactly and formally, as follows. Minimize

\[
J(w, A, B, C) = \frac{1}{2} \int_0^T \sum_{p=1}^P C_p dt
\]

(2.9)

where \(C_p = \|z_p - d_p\|^2 + \lambda \left\| \frac{\partial g}{\partial y_p} \frac{\partial^2 y_p}{\partial y_p \partial u_p} \right\|^2\) for the optimization problem 2.4 through 2.7 and \(C_p = \left\| \frac{\partial g}{\partial y_p} \frac{\partial^2 y_p}{\partial y_p \partial u_p} \right\|^2\) in the case of learning task 2.5 through
Improving Generalization Capabilities of Dynamic Neural Networks

2.8 subject to constraints

\[ \dot{y}_p = f(y_p, w, u_p) \]
\[ \frac{d}{dt} \left( \frac{\partial y_p}{\partial u_p} \right) = \frac{df}{du_p} \]
\[ \frac{d}{dt} \left( \frac{\partial^2 y_p}{\partial^2 u_p} \right) = \frac{d^2 f}{du_p^2} \]

(2.10)

\[ y_p(0) = A, \quad \frac{\partial y_p}{\partial u_p}(0) = B, \quad \frac{\partial^2 y_p}{\partial^2 u_p}(0) = C \]

(2.11)

and

\[ \frac{1}{2} \int_0^T \sum_{p=1}^L D_p dt \leq 0, \]

(2.12)

where \( D_p = 0 \) for the optimization problem 2.4 through 2.7 and \( D_p = \|z_p - d_p\|^2 - \frac{2\epsilon}{T} \) for the optimization task 2.5 through 2.8, respectively; \( 0 < \epsilon \) is a small given number (accuracy of learning). The inequality 2.12 takes into account (in the nontrivial case) the condition of tracking the desired trajectories \( d \) by the neural network outputs \( y \) (a conventional task most commonly used in the literature). Let us note that giving some value for \( \epsilon \) is more natural than searching for a compromise (i.e., \( \lambda \)) between terms \( \|z_p - d_p\|^2 \) and \( \| \frac{\partial g}{\partial y_p} \| \) in performance index 2.4. In addition, the generalization error is also proportional to an \( \epsilon \) for finite time sequences (see, e.g., DasGupta & Sontag 1996, and Hammer, 1997 for details). One constructive method of determining the weight vector \( w \) that fulfills the constraint 2.12 has been proposed by Galicki et al. (1999). Expressions 2.9 through 2.12 present a relatively complex learning problem whose solution by means of classical networks (with constant weights) seems to be difficult (or even impossible). The reason is that the network should now accomplish simultaneously more tasks than one, that is, minimize proper performance index 2.9 by maintaining the conventional task given by equation 2.12. Therefore, more parameter redundancy is required for the network to fulfill all the constraints 2.10 through 2.12. Generalized dynamic neural networks (GDNNs), first introduced by Leistritz, Galicki and Witte (1998) and Galicki et al. (1999) offer a possibility of tackling this problem. They are defined by introducing time-dependent weights into a Hopfield-type network. As was shown by Galicki et al. (1999), it is possible to find a weight vector \( w \) such that the constraint 2.12 is satisfied for each \( \epsilon \geq 0 \). This property makes the learning algorithm that is presented in the next section computationally efficient.
The time-dependent weights may be treated as controls. Thus dependences 2.9 through 2.12 formulate the learning task as an optimal control problem. The fact that there exist constraints imposed on coordinates of vector $y_p$ (state equality and inequality constraints) makes its solution difficult. An approach based on the application of the variational principle of Pontryagin's maximum principle will be proposed to solve the optimization problem.

3 Learning by Smoothing the Network Mapping

Before presenting the algorithm, all the variables $y_p, \frac{\partial y_p}{\partial u_p}, \frac{\partial^2 y_p}{\partial u_p^2}$ will be transformed into a vector form, which is more convenient for further considerations. Therefore, the following state vectors are introduced:

$$x_p = (x_{p,1}, \ldots, x_{p,N}^{N+2+N^3})^T = \left( y_{p,1}, \ldots, y_{p,N}, \frac{\partial y_{p,1}}{\partial u_{p,1}}, \ldots, \frac{\partial y_{p,N}}{\partial u_{p,N}}, \frac{\partial^2 y_{p,1}}{\partial u_{p,1}^2}, \ldots, \frac{\partial^2 y_{p,N}}{\partial u_{p,N}^2} \right)^T$$

and

$$x_0 = (A_1, \ldots, A_N, B_{11}, \ldots, B_{NN}, C_{111}, \ldots, C_{NNN})^T, \quad p = 1: P.$$ 

Using the quantities $x_0, x_p$ and taking into account expressions 2.1, 2.5, and 2.6, the dynamic equations of the neural network may be written in an extended state space as

$$\dot{x}_p = F(x_p, w, u_p) \quad (3.1)$$

where

$$F = \left( f_1, \ldots, f_p, \frac{df_1}{du_{p,1}}, \ldots, \frac{df_n}{du_{p,N}}, \frac{\partial^2 f_1}{\partial u_{p,1}^2}, \ldots, \frac{\partial^2 f_n}{\partial u_{p,N}^2} \right)^T.$$ 

Consequently, the optimal control problem given by expressions 2.9 through 2.12 is transformed into a functional form (which is equivalent to the previous one), as follows. Minimize

$$J(w, x_0) = \int_0^T C dt \quad (3.2)$$

where

$$C = \sum_{p=1}^P \left( \frac{1}{2} \|(x_{p,1}, \ldots, x_{p,L})^T - d_p\|^2 + \frac{1}{2} \lambda \sum_{m=N+2+N^2}^{N+2+N^3} (x_{p,m})^2 \right)$$
for the optimization problem 2.4 through 2.7 and $C = \sum_{p=1}^{P} \frac{1}{2} \sum_{m=N+N^2+1}^{N^2+LN^2} (x_{p,m})^2$ in the case of learning task 2.5 through 2.8, subject to the constraints

$$
\dot{x}_p = F(x_p, w, u_p)
$$

(3.3)

$$
x_p(0) = x_0
$$

(3.4)

$$
G(w, x_0) \leq 0
$$

(3.5)

where $G = 0$ for the optimization problem 2.4 through 2.7 and $G = \int_{t_0}^{T} \sum_{p=1}^{P} \frac{1}{2} \sum_{m=N+N^2+1}^{N^2+LN^2} (x_{p,m})^2$ in the case of learning task 2.5 through 2.8, respectively. Due to state constraints 3.4 and 3.5, the application of the variational formulation of Pontryagin's maximum principle (Fedorenko, 1978) is proposed here to solve the optimal control problem 3.2 through 3.5 (see, e.g., Galicki, 1998, for details regarding the variational formulation of Pontryagin's maximum principle). This formulation requires a specification of the initial weight vector $w^0 = w^0(t)$, $t \in [0, T]$. However, $w^0$ does not necessarily satisfy the state inequality constraint 3.5. The initial trajectories $x_i^0(t)$ are obtained by solving equation 3.3 for each $u_p$, $p = 1 : P$. By assumption, $w^0$ and $x_0$ do not minimize the performance index, equation 3.2. The use of the variational formulation of Pontryagin's maximum principle requires incrementation of functionals 3.2 and 3.5. Therefore, it is assumed that the weight vector $w^0 = w^0(t)$ and initial state $x_0$ are perturbed by small variations $\delta w = \delta w(t) = (\delta w_m : 1 \leq m \leq N^2)$ and $\delta x_0 = (\delta x_{1,0}, \ldots, \delta x_{N+N^2+\nabla, 0})^T$, where $||\delta w||_\infty = \max |\delta w_m| \leq \rho$; $\|\delta x_0\|_\infty = \max \{ \|\delta x_{i,0}\| \} \leq \xi$; $\rho$ and $\xi$ are given small numbers ensuring the correctness of the presented method. According to the theory of small perturbations (Pontryagin et al., 1961) and on account of the fact that $\delta x_0(t) = \delta x_0$, the value of the functional $J$ for the perturbed weight vector $w^0 + \delta w$ and initial state $x_0 + \delta x_0$ may be expressed to the first order (by neglecting the higher-order terms $o(\delta w, \delta x_0)$) by the following dependence,

$$
J(w^0 + \delta w, x_0 + \delta x_0) = J(w^0, x_0) + \delta J(w^0, x_0, \delta w, \delta x_0),
$$

(3.6)

where $\delta J(w^0, x_0, \delta w, \delta x_0) = \sum_{p=1}^{P} \int_{t_0}^{T} \left((F_w(p, t))^T \Psi_p, \delta w\right) d\tau + \left(\Psi_p(0), \delta x(0)\right)$ is the Frechet differential of the functional $J(\cdot, \cdot)$; $\Psi_p(\cdot)$ stand for conjugate mappings, which are determined by the solution of the Cauchy problem:

$$
\dot{\Psi}_p + (F_x(t))^T \Psi_p = -C_x(t); \quad \Psi_p(T) = 0;
$$

$$
F_w(p, t) = \frac{\partial F}{\partial w} \bigg|_{x=x_0^0, w=w_0^0}; \quad F_x(t) = \frac{\partial F}{\partial x_p} \bigg|_{x=x_0^0, w=w_0^0};
$$
The value of the functional determined by the left-hand side of equation 3.5 for weights $w^0 + \delta w$ and perturbed initial state $x_0 + \delta x_0$ is given to the first order, as follows,

$$G(w^0 + \delta w, x_0 + \delta x_0) = G(w^0, x_0) + \delta G(w^0, x_0, \delta w, \delta x_0),$$

(3.7)

where $\delta G(w^0, x_0, \delta w, \delta x_0) = \sum_{p=1}^{P} \int_{t_0}^{T} (F_p(x(t), t))^T \Phi_p, \delta w) dt + (\Phi_p(0), \delta x(0))$ is the Frechet differential of the functional $G(\cdot, \cdot)$; mappings $\Phi_p(\cdot)$ are computed based on the solution of the following differential equations:

$$\dot{\Phi}_p + (F_p(t))^T \Phi_p = -D_p(x(t)); \quad \Phi_p(T) = 0; \quad D_p(x(t)) = \left( \frac{\partial D_p}{\partial x_p} \right)_{x = x_0^p}.$$

For properly selected variations $\delta w$ and $\delta x_0$, the Frechet differentials of functionals 3.2 and 3.5 approximate the increments of these functionals with any desired accuracy. Consequently, the variational formulation of Pontryagin’s maximum principle takes the following form,

$$J(w^0 + \delta w^*, x_0 + \delta x_0^*) = J(w^0, x_0) + \min_{(\delta w, \delta x_0)} \{ \delta J(w^0, x_0, \delta w, \delta x_0) \},$$

(3.8)

subject to the constraints

$$G(w^0, x_0) + \delta G(w^0, x_0, \delta w^*, \delta x_0^*) \leq 0$$

$$\|\delta w^*\|_{\infty} \leq \rho$$

$$\|\delta x_0^*\|_{0, \infty} \leq \xi.$$  

(3.9)

The assumption of nonoptimality of weight vector $w^0$ and initial network state $x_0$ implies the existence of sufficiently small variations $\delta w^*$ and $\delta x_0^*$ such that $J(w^0 + \delta w^*, x_0 + \delta x_0^*) < J(w^0, x_0)$. Consequently, the task of minimizing functional 3.8 is well posed, and variations $\delta w^*$, $\delta x_0^*$ are results of its solution.

A finite-dimensional approximation of the optimal control problem defined by dependences 3.8 and 3.9 seems to be very effective for solving it numerically. The process of approximation for weight variation $\delta w$ may be accomplished, for example, by using quasi-constant or quasi-linear functions or splines. This article looks for a sequence of weight variations in a class of polynomials

$$\delta w(t) = VC(t),$$

(3.10)

where $V \in \mathbb{R}^{N^2 \times M}$ denotes the matrix of coefficients to be found; $M$ is a given number of basic functions (accuracy of computations); and $C(t)$ stands for
Improving Generalization Capabilities of Dynamic Neural Networks

the \( M \)-dimensional vector of specified basic functions (e.g., Chebyshev’s polynomials scaled to \([0, T]\)). Let us note that \( M \) could be optimized (implying a structural optimization), resulting in an increase of computational efficiency (i.e., finding a minimal number of basic functions to give a satisfactory minimum of functional 3.5). For simplicity, \( M \) is assumed to be given. Consequently, only weight variation (coefficient matrix \( V \)) and initial network state variation \( \delta x_0 \) are subject to the minimization process. In this way, the determination of optimal variations \( \delta w^* \) and \( \delta x_0^* \) from dependences 3.8 and 3.9 is reduced to solving a linear programming problem with the following cost function,

\[
J(w^0 + \delta w, x_0 + \delta x_0) = J(w^0, x_0) + \int_0^T \left( \sum_{p=1}^P (Fw(p, t))^T \Psi_p, VC(t) \right) dt + \left( \sum_{p=1}^P \Psi_p(0), \delta x_0 \right),
\]

(3.11)

with respect to \( V \) and \( \delta x_0 \), subject to the constraints

\[
G(w^0, x_0) + \int_0^T \left( \sum_{p=1}^P (Fw(p, t))^T \Phi_p, VC(t) \right) dt + \left( \sum_{p=1}^P \Phi_p(0), \delta x_0 \right) \leq 0
\]

\[
\|V\|_\infty \leq \rho'
\]

\[
\|\delta x_0\|_\infty \leq \xi,
\]

(3.12)

where \( \rho' \) is a given small number (step length). After the determination of optimal variations \( \delta w^* \), \( \delta x_0^* \) from the minimization task 3.11–3.12, the computations are repeated for the revised quantities \( w^1 = w^0 + \delta w^* \) and \( x_0^1 = x_0 + \delta x_0^* \). A sequence of pairs \( (w^k, x_0^k) \), \( k = 0, 1, \ldots, x_0^0 = x_0 \) may thus be obtained as a result of solving the iterative approximation scheme 3.11–3.12. Each element of this sequence corresponds to the state trajectory \( \chi^k_0(\cdot) \). The continuity of \( D_p \) with respect to \( x_p \) and the inequality \( \max_{k,p} \int_0^T D_p(x_p^0)dt < \infty \) yields the following relation, \( \sum_{p=1}^P \int_0^T D_p(x_p^0)dt \leq 0 \), where \( x_p^0 \) stands for a limit trajectory whose existence is theoretically ensured. Consequently, the convergence of this sequence to an optimal solution (in general local) follows from the work of Galicki et al. (1999). In order to run a minimization procedure 3.11–3.12, initial weights \( w^0 \) and state vector \( x_0 \) should be
specified. As is known, GDNNs render it possible to efficiently find such weights that fulfill state inequality constraints 2.12. Next, initial vectors \( x_0 \) and \( w^0 \) are iteratively modified using variations \( \delta x_0, \delta w(t) = V C(t) \) in order to carry out a proper minimization of the performance index 3.11. The generality of the iterative scheme 3.11–3.12 carries with it a penalty; somewhat larger computations may be required during the learning process. They may be significantly reduced by taking into account the square norm of network sensitivity instead of the mapping curvature. The role of the sensitivity is also to smooth out the rapid variations in the network mapping. As compared to the curvature, the network sensitivity is more local (the first derivative used instead of the second one), and thus the generalization capability of the network may become somewhat worse for a small learning set, as the numerical results presented in the next section show. Nevertheless, application of the sensitivity measure as the penalty term leads to the minimization of the following performance index:

\[
J = \frac{1}{2} \int_0^T \left( \sum_{p=1}^P \| z_p - d_p \|^2 + \lambda \left[ \frac{\partial g}{\partial y_p} \frac{\partial y_p}{\partial u_p} \right]^2 \right) dt. \tag{3.13}
\]

Let us note that minimizing term

\[
\left[ \frac{\partial g}{\partial y_p} \frac{\partial y_p}{\partial u_p} \right]^2
\]

forces the network output to be locally invariant with respect to small perturbations of external input \( u_p(t) \) at time instant \( t \). In such a case, state vector \( x_p \) and the network mapping \( F \) take the following form,

\[
x_p = (x_{p,1}, \ldots, x_{p,N+N^2})^T = \left( y_{p,1}, \ldots, y_{p,N}, \frac{\partial y_{p,1}}{\partial u_{p,1}}, \ldots, \frac{\partial y_{p,N}}{\partial u_{p,N}} \right)^T \tag{3.14}
\]

where \( x_0 = (A_1, \ldots, A_N, B_{11}, \ldots, B_{NN})^T \), and

\[
F = \left( f_1, \ldots, f_p, \frac{df_1}{du_{p,1}}, \ldots, \frac{df_N}{du_{p,N}} \right)^T. \tag{3.15}
\]

Let us note that using performance index 3.13 results in a significant decrease of the size of the state vector (by \( N^3 \)). Consequently, iterative scheme 3.11–3.12 is now reduced as follows. Minimize

\[
J(w^0, x_0) + \int_0^T \left( \sum_{p=1}^P (F_w(p, t))^T \Psi_p, V C(t) \right) dt + \left( \sum_{p=1}^P \Psi_p(0), \delta x_0 \right) \tag{3.16}
\]
with respect to $V$ and $\delta x_0$, subject to the constraints

$$\|V\|_\infty \leq \rho'$$

$$\|\delta x_0\|_\infty \leq \xi.$$  \hspace{1cm} (3.17)

As is easy to see, the above linear programming problem may be solved analytically, which significantly reduces the computational cost.

Eliminating regularization parameter $\lambda$ from equation 3.13 leads (similarly as in the case of network curvature) to the minimization of the following performance index,

$$J = \frac{1}{2} \int_0^T \sum_{p=1}^P \left\| \frac{\partial g}{\partial y_p} \frac{\partial y_p}{\partial u_p} \right\|^2 dt.$$  \hspace{1cm} (3.18)

subject to constraints

$$y_p(0) = A, \quad \frac{\partial y_p}{\partial u_p}(0) = B,$$  \hspace{1cm} (3.19)

$$\frac{1}{2} \int_0^T \sum_{p=1}^P \|z_p - d_p\|^2 dt \leq \epsilon.$$  \hspace{1cm} (3.20)

Application of iterative minimization scheme 3.11–3.12 with $x_p$ and $F$ defined by equations 3.14 and 3.15 yields an optimal solution.

It follows from the considerations carried out that a relatively general method for training a continuous-time recurrent neural network subject to its good generalization is proposed here. However, the generality of our approach carries with it a penalty: somewhat larger computations are required to find an optimal solution. In such a context, determination of time-dependent network curvature $\frac{\partial^2 y_p}{\partial u_p^2}$ requires $O(N^3)$ solutions of ODEs, which would seemingly mean that the proposed method would be appropriate only for small problems. Nevertheless, the time-dependent weights result in an overparameterization of such networks. Consequently, this property gives the possibility of reducing the network structure (to decrease the number of neurons $N$) even for the relatively complex problems considered in section 4. Moreover, as the numerical simulations (carried out in the next section) show, the sensitivity term $\frac{\partial y_p}{\partial u_p}$ (requiring only $O(N^2)$ solutions of ODEs, which significantly reduces the computational burden) has similar results compared to those obtained by minimizing the norm of network curvature. However, minimizing the norm of network curvature needs smaller training sets (which may be sometimes practically desirable especially in the medical field) as compared to minimizing the norm of sensitivity. This
property seems to be obvious since the second derivative carries with it more information about the (local) network behavior as compared to the first derivative.

4 Computer Examples

The purpose of this section is to show, on the basis of several chosen tasks, the essential improvement of the generalization capability of GDNNs trained with the algorithms presented in section 3. Particular forms of performance indexes used in computer examples are dictated by the necessity of making it possible to compare the obtained numerical results of the methods proposed in section 3 with training methods without regularization (classical performance index) known from the literature.

4.1 One-Class Task. The first task is the training of a circular trajectory by means of a fully connected GDNN with three neurons. In fact, the network state evolution is governed by the following differential equations,

\[
\dot{y} = \begin{pmatrix}
-\gamma_1 + \arctan(\sum_{j=1}^{3} \gamma_j \cdot y_j) \\
-\gamma_2 + \arctan(\sum_{j=1}^{3} \gamma_{j+3} \cdot y_j) \\
-\gamma_3 + \arctan(\sum_{j=1}^{3} \gamma_{j+6} \cdot y_j) + u_3
\end{pmatrix},
\]

(4.1)

with the initial network state \(y(0) = (0, 0, 0)^T\) and a network evolution time of \(T = 12\pi\). The network output \(z\) is defined by the first two coordinates:

\[
z(t) = \begin{pmatrix}
y_1(t) \\
y_2(t)
\end{pmatrix}.
\]

Evidently, \(y(0)\) is a fix point of equation 4.1 if no external input \((u_3(t) = 0)\) is provided to the network. That is why a nonzero input \(u_3\) is necessary in order to produce the trajectory

\[
d(t) = \begin{pmatrix}
\cos(t) \\
\sin(t)
\end{pmatrix}.
\]

Consequently, a constant input function \(u_3(t) = 1\) was used in this example. In fact, the condition \(u_3(t) \neq 0\), \(0 \leq t \leq \delta\), for an arbitrary \(\delta > 0\) is sufficient for the solution of the learning task (Galicki et al. 1999). In other words, the principal task of the external input is to thrust aside the system 4.1 from \(y(0)\).

Starting with a common initial weight vector \(w^0 = 0\) for all \(t\), nine GDNN were trained: the first one (net1) by means of the classical performance index \((\lambda = 0)\); four networks (net2–net5) by means of the performance index, equation 3.13 including a first-order regularization term \(\lambda \in \{0.02, 0.05, 0.1, 0.2\}\); and four further networks (net6–net9) by means of
the performance index, equation 2.4 including a second-order regulariza-
tion term $\lambda \in \{0.02, 0.05, 0.1, 0.2\}$. Defining the performance contributions

$$J_0 = \int_{2\pi}^{12\pi} \|z - d\|^2 dt, \quad J_1 = \int_{2\pi}^{12\pi} \left\| \frac{\partial g}{\partial y} \frac{\partial y}{\partial u} \right\|^2 dt,$$

$$J_2 = \int_{2\pi}^{12\pi} \left\| \frac{\partial g}{\partial y} \frac{\partial^2 y}{\partial u^2} \right\|^2 dt,$$

one can see the expected influences of the regularization techniques in Figure 1. It can be seen from this figure that the regularization technique does not influence error $J_0$ in dependence on $\lambda$. Moreover, all networks have similar approximation properties. In contrast, the smoothing terms $J_1$ and $J_2$ show a direct dependence on the regularization constant. Moreover, a regularization with $J_1$ leads to a decrease of $J_2$, and vice versa in this example. This observation is a typical one in the presence of a single training pattern ($P = 1$) but cannot be affirmed for $P > 1$.

In order to evaluate the generalization capabilities of the networks, the network outputs with respect to disturbed external inputs were considered.

Figure 1: Performance partitions for trained networks: (A) the classical error, (B) sensitivity, and (C) curvature.
At first, gaussian noise with different variances was added to \( u_3 \), where

\[
    u_{3,\sigma} = u_3(t) + \sigma_a \cdot X(t), \quad X(t) \sim N(0, 1), \quad \sigma_a = 0.02, 0.04, \ldots, 2
\]
denotes the noisy input. Second, the input \( u_3 \) was modified by random Chebyshev’s polynomials of order 11 (scaled to \([0, 12\pi]\)). That is, the perturbed input was given by

\[
    u_{3,\sigma} = u_3(t) + \sigma_b \cdot Y(t), \quad Y(t) = \sum_{i=0}^{10} \eta_i \cdot C_i(t), \quad \sigma_b = 0.02, 0.04, \ldots, 2
\]

with a set of independent identical uniform distributed random variables \( \eta_i \sim U([-1, 1]) \). At each \( \sigma \), 100 realizations for both types of perturbations were used as external inputs and provided to all of the nine trained networks. The mean L2 distances between the network response and the target are given in Figure 2. Obviously, the regularization with \( J_1 \) or \( J_2 \) improves the generalization capabilities, where the regularization with the first-order term \( J_1 \) is more favorable due to the better cost-value ratio.

### 4.2 Two-Classes Task

The second example deals with the classification of external inputs, which are defined as solutions of the one-dimensional van der Pol equation,

\[
    \ddot{u}_\mu = \mu \cdot (1 - u_\mu^2) \dot{u}_\mu - u_\mu, \quad (4.2)
\]

for any real number \( \mu \). By means of the parameter \( \mu \), we define two classes \( \omega_1 \) and \( \omega_2 \) distributed according to the class conditional density functions (see Figure 3)

\[
    \varphi_1(\mu) = \frac{1}{1.2 \cdot \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\mu}{1.2} \right)^2}
\]

and

\[
    \varphi_2(\mu) = \frac{1}{2} \cdot \frac{1}{0.8 \cdot \sqrt{2\pi}} \left( e^{-\frac{1}{2} \left( \frac{\mu + 0.4}{0.8} \right)^2} + e^{-\frac{1}{2} \left( \frac{\mu - 0.4}{0.8} \right)^2} \right).
\]

Assuming equal a priori probabilities for the classes \( \omega_1 \) and \( \omega_2 \), the optimal thresholds for the discrimination are given by the points of intersection of \( \varphi_1 \) and \( \varphi_2 \) (Fukunaga, 1990). In fact, they are approximately equal to \( \pm 1.558 \) and \( \pm 7.082 \). However, for our further considerations, the values \( \pm 7.082 \) are disregarded, since the corresponding contribution to the classifier error is less than \( 10^{-8} \).

By this setup, we have two classes of functions whose distribution is implicitly known by the parameter \( \mu \). The question now is whether a GDNN
is able to discriminate the two classes without any preprocessing or estimation of the underlying parameter $\mu$ and which sample size is sufficient to solve this problem with different performance indexes. First, a random training sample of size 84 (42 from $\omega_1$, 42 from $\omega_2$) was generated. Furthermore, in order to investigate the influence of the sample size, 10 subsamples of sizes 12, 20, ..., 84 were selected. For each subsample, three GDNNs were trained: one by means of the classical performance index ($\lambda = 0$), one by means of the performance index 3.13 (first-order regularization) with $\lambda = 5 \cdot 10^{-2}$, and one by means of the performance index 2.4 (second-order regularization), where $\lambda = 25 \cdot 10^{-4}$. The common topology for all fully connected GDNNs with three neurons is given by

$$\dot{y} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{pmatrix} = \begin{pmatrix} -y_1 + \arctan(\sum_{j=1}^{3} w_j \cdot y_j) \\ -y_2 + \arctan(\sum_{j=1}^{3} w_{j+3} \cdot y_j) \\ -y_3 + \arctan(\sum_{j=1}^{3} w_{j+6} \cdot y_j) + u_\mu \end{pmatrix},$$

with $y(0) = (0, 0, 0)^T$ and a network evolution time of $T = 10$. The network input $u_\mu$ is defined with starting conditions $u_\mu(0) = 2$ and $\dot{u}_\mu(0) = 0$. The GDNN output $z$ is defined by the first coordinate. The networks were
Figure 3: Class conditional densities. The solid line stands for the class $\omega_1$ and the dashed one for $\omega_2$. The dotted lines indicate the optimal thresholds assuming equal a priori probabilities for both classes.

forced to produce an output $d_1(t) = 0.1 \cdot t$ if an input $u_\mu$ with $\mu \in \omega_1$ was provided, or $d_2(t) = -0.1 \cdot t$ in the case that $\mu \in \omega_2$ holds. Starting from this construction, the determination of the classifier decision rule is straightforward:

$$\int_0^{10} (z - d_1)^2 dt < \int_0^{10} (z - d_2)^2 dt \Rightarrow \omega_1$$

$$\int_0^{10} (z - d_1)^2 dt > \int_0^{10} (z - d_2)^2 dt \Rightarrow \omega_2.$$

The estimated classifier errors for all GDNNs are depicted in Figure 4. It can be stated that even a small sample size of $n = 36$ is sufficient for the construction of a GDNN-based classifier, independent of the error functional used for the network training. Smaller sample sizes lead to the well-known overfitting effect using the classical performance index. In contrast, a smoothing of the network mapping can reduce the overfitting and results in high-performance classifiers for a sample size of 28. For smaller samples, the regularization cannot eliminate the overfitting, but it intensely reduces it and, for this reason, the classifier errors too. The higher value for $n = 20$
is most probably attributed to the local convergence of the training procedure, albeit the second-order regularization term seems to be the most capable tool for small samples. For sufficiently large sample sizes, there is no need to work with penalty terms, subject to the condition that an appropriate network topology was chosen. However, the regularization does not influence the classifiers negatively.

4.3 Real-Life Example. The last example is given by a classification task of gait analysis data of children with cerebral palsy, a neuromuscular disorder that occurs in about 3 of 1000 live births. A permanent but not progressive brain lesion is frequently associated with spasticity and deficiencies in voluntary movement control. An excessive plantar flexion at the ankle joint is frequently observed in ambulatory children with cerebral palsy. The deformity results in an awkward gait and interferes with balance during standing and walking. Whenever this excessive flexion becomes rigid, it represents an indication for orthopedic surgery to lengthen the calf musculature or its tendon. To date, the rigidity of the calf muscles had to be tested under general anesthesia because spasticity and increased muscle tone in

Figure 4: Estimated classifier errors for all trained GDNN. The dotted lines indicate the Bayes error ($\approx 0.1702$) estimated directly from $\phi_1$ and $\phi_2$ for equal a priori probabilities.

Downloaded from http://www.mitpressjournals.org/doi/pdf/10.1162/089976604773717603 by guest on 31 May 2021
patients with cerebral palsy make physical examination difficult and any clinical test unreliable.

While treatment options for the two types of deformity have been studied extensively, the distinction between dynamic tightness and a fixed contracture remains a clinical challenge. To date, the evaluation of ankle dorsiflexion under general anesthesia and muscle relaxation still resembles the most accurate but most often infeasible clinical method to assess the quality of this deformity. Instrumented gait analysis has considerably contributed to the understanding of gait dysfunction in cerebral palsy. Also, the technique has proven to be a valuable tool to objectively document gait changes after surgical interventions at the calf for gait improvement. However, as a diagnostic tool for discriminating between dynamic tightness and fixed contracture of the triceps surae, instrumented gait analysis is not yet established.

Instrumented gait analysis provides a tool to assess human walking three-dimensionally and to document gait abnormalities in children with spastic disorders. The system uses a video-based six-camera motion-capturing device and up to 18 reflective markers that are placed at various landmarks of the lower limbs. These markers are then tracked in space to provide motion paths of the involved limb segments during walking. Finally, kinematic segment parameters and angle changes at joints connecting individual segments can be computed. Data acquisition consists of recording at least five valid walking cycles while patients are asked to walk at a self-selected speed over an 8-meter walkway. The aim of this example was to investigate whether GDNNs are able to differentiate between rigid (the first class) and flexible (the second class) plantar flexion deformity in children with spastic cerebral palsy. The signals used as external inputs were given by the time courses of ankle angle $\theta_{aan}$, ankle moment $\tau_{am}$, knee angle $\theta_{unakan}$, and knee moment $\tau_{km}$ (see Figure 5) during the walking of 56 children (22 rigid, 34 flexible). The corresponding GDNN is defined by

$$\dot{y} = \left(\begin{array}{c}
\dot{y}_1 \\
\dot{y}_2 \\
\dot{y}_3 \\
\dot{y}_4 \\
\dot{y}_5
\end{array}\right) = \left(\begin{array}{c}
-y_1 + \arctan(\sum_{j=1}^{5} w_j \cdot y_j) \\
-y_2 + \arctan(\sum_{j=1}^{5} w_{j+5} \cdot y_j) + \frac{\theta_{aan}}{20} \\
-y_3 + \arctan(\sum_{j=1}^{5} w_{j+10} \cdot y_j) + \tau_{am} \\
-y_4 + \arctan(\sum_{j=1}^{5} w_{j+15} \cdot y_j) + \frac{\theta_{ukan}}{40} \\
-y_5 + \arctan(\sum_{j=1}^{5} w_{j+20} \cdot y_j) + \tau_{km}
\end{array}\right),$$

with $y(0) = (0, 0, 0, 0, 0)^T$, a network evolution time of $T = 1$ and target functions $\tilde{d}_p(t) \in \{0.5 \cdot t, -0.5 \cdot t\}$, where $\tilde{d}_p(t) = 0.5 \cdot t$ was taken for the first ("rigid") class and $\tilde{d}_p(t) = -0.5 \cdot t$ for the second ("flexible") one. The GDNN output $z$ is defined by the first coordinate, $z = y_1$.

From a practical point of view, it is suitable to discriminate between neurons provided with external inputs and the neurons defining the network output. Consequently, the network topology used in this example was chosen to be as small as possible (four input neurons and one output). The time
courses of angles and moments were of different scales. Due to technical reasons, the angles had to be scaled by 20 and 40, respectively.

In order to access the generalization capabilities, the complete data set was randomly split into training sets of sizes 23, 18, and 13 and their corresponding independent test sets of sizes 33, 38, and 43. For each training set, three GDNNs were trained according to the performance indices given in Table 1. Table 2 depicts the classifier performances in dependence on different training sets and the various performance indices used for the network training. Similar to the previous example, the smoothing of the net-
work mapping improves the generalization capabilities. Furthermore, the method proposed in section 3 performed better than the classical training algorithm in all cases. As in the first example, the smoothing by means of the first-order term is more favorable due to the better cost-value ratio. For comparable purposes, the clinical examination without anesthesia results in a classification rate of 39.3%, and a subjective chart analysis of gait analysis data by an experienced clinical expert yielded 80.4% correct classifications.

We have seen that the smoothing of the network mapping can improve the generalization capabilities of GDNNs. Thereby, it is irrelevant whether the regularization technique or the constraint learning is applied. However, it is considerably easier to fix the bound $\epsilon$ instead of the regularization constant $\lambda$ from a practical point of view. The reason is that one has to know the order of magnitudes of the error term $J_0$ and the regularization term $J_1$ or $J_2$ in order to find an appropriate compromise between learning quality and network mapping smoothing. In the case of constraint learning, only information about $J_0$ is necessary. For example, this information can be easily extracted from a few classical training runs.

4.4 The Time-series Prediction. The purpose of this section is to show on a chosen test bed time-series prediction problems (sunspot prediction and chemical process control) the performance of GDNNs trained with the algorithm proposed in section 3.

4.4.1 Wolfe’s Sunspot Data. A data set that has been frequently (probably most often) investigated for time-series prediction is the set of Wolfe’s sunspot data. The time series consists of the average number of sunspots observed per year from 1700 to the present.

In order to keep comparability to other approaches, samples recorded until 1920 were used for training and data from years 1921–1955 (test set 1) and 1956–1979 (test set 2) for testing.

GDNN-based predictors were constructed by fully connected GDNNs consisting of six neurons with the common neural activation function $\alpha(z) = (1 + e^{-z})^{-1}$ and constant weights. Let $\kappa_{SP}$ be the sunspot time series. Then
Table 3: Root-Mean-Squared Errors for Different Training Performance Indexes.

<table>
<thead>
<tr>
<th></th>
<th>Test Set 1</th>
<th>Test Set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical training</td>
<td>0.0688</td>
<td>0.1404</td>
</tr>
<tr>
<td>Smoothed network mapping</td>
<td>0.0642</td>
<td>0.1292</td>
</tr>
<tr>
<td>With regularization</td>
<td>0.0774</td>
<td>0.1518</td>
</tr>
</tbody>
</table>

The network inputs for the prediction of \( \kappa_{SP}(t) \) were defined by

\[
\mathbf{u}(t) = (\kappa_{SP}(t - 1), \kappa_{SP}(t - 2), \kappa_{SP}(t - 4),
\kappa_{SP}(t - 6), \kappa_{SP}(t - 8))^T / 95.1 - 1,
\]

and the desired network output was given by

\[
d(t) = \kappa_{SP}(t) / 190.2.
\]

In order to investigate the influence of the constraint learning, three performance indexes were used for the network training: the classical one, the first-order term (see equation 3.18) for smoothing the network mapping, and the first-order regularization term according to equation 3.13 with \( \lambda = 10^{-3} \).

Table 3 depicts the prediction error as a root-mean-squared error between the GDNN-based prediction and Wolfe’s sunspot data. As is seen from Table 3, a regularization with the penalty term 3.13 worsens the performance of the network as compared to minimization of pure sensitivity term 3.18 with constraint 2.12 (we ran the method presented in this article with other values for \( \lambda \), but they did not yield better results). Moreover, our results are comparable with the best results known from the literature (Rementeria & Olabe, 2000; Wan, 1997; Azoff, 1993).

4.4.2 Continuous Stirred Tank Reactor. The next example is concerned with a chemical process modeling problem: the modeling of pH in a continuous stirred tank reactor (CSTR) (Bhat & McAvoy, 1990). The considered CSTR has two input streams—one containing sodium hydroxide (NaOH) and the other one acetic acid (HAC). Every stream has a flow rate \( F_k \) and a concentration \( C_k \), \( k = 1, 2 \), where the index 1 stands for HAC, and index 2 points to the values of NaOH. Let \( \xi_1 \) be the notation for the total acetate concentration [HAC + AC\(^-\)], and let \( \xi_2 \) be the concentration of Na\(^+\). Assuming that the acid-base equilibrium and electroneutrality relationships hold, the total acetate balance and the sodium balance are given by Bhat and McAvoy (1992):

\[
V \cdot \dot{\xi}_1 = F_1 C_1 - (F_1 + F_2) \cdot \xi_1
\]
Table 4: Parameters and Initial Values for CSTR.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V$</td>
<td>Tank volume</td>
<td>1000 liter</td>
</tr>
<tr>
<td>$F_1$</td>
<td>Flow rate of acid</td>
<td>81 l/min</td>
</tr>
<tr>
<td>$F_2$</td>
<td>Flow rate of base</td>
<td>515 l/min</td>
</tr>
<tr>
<td>$C_1$</td>
<td>Concentration of acid in $F_1$</td>
<td>0.32 mole/liter</td>
</tr>
<tr>
<td>$C_2$</td>
<td>Concentration of base in $F_2$</td>
<td>0.05 mole/liter</td>
</tr>
<tr>
<td>$K_a$</td>
<td>Acid equilibrium constant</td>
<td>$1.76 \times 10^{-5}$</td>
</tr>
<tr>
<td>$K_w$</td>
<td>Water equilibrium constant</td>
<td>$1.00 \times 10^{-14}$</td>
</tr>
<tr>
<td>$\zeta_1(0)$</td>
<td>Initial concentration of acid</td>
<td>0.0435 mole/liter</td>
</tr>
<tr>
<td>$\zeta_2(0)$</td>
<td>Initial concentration of base</td>
<td>0.0432 mole/liter</td>
</tr>
</tbody>
</table>

and

$$V \cdot \dot{\zeta}_2 = F_2 C_2 - (F_1 + F_2) \cdot \zeta_2.$$ 

The pH is calculated from the previous equations by

$$[H^+]^3 + (K_a + \zeta_2) \cdot [H^+]^2 + (K_a \cdot \zeta_2 - K_a \cdot \zeta_1 - K_w) \cdot [H^+] - K_a K_w = 0$$

and

$$\text{pH} = -\log [H^+]$$

with process parameters given in Table 4 (steady-state operating point; Bhat & McAvoy, 1990). The objective is the prediction of pH in dependence on changes in the flow rate of sodium stream $F_2$. For comparison, the same parameters as in Sridhar Bartlett, and Seagrave (1998) were selected. In fact, the sampling time was 0.2 min, and the training set was designed over 1 hour by uniform distributed random flow rates $F_2$ within 10% of the steady-state operation point. Three further hours were used for testing.

Two GDNN-based predictors were constructed by fully connected GDNNs consisting of 20 neurons with the common neural activation function $10 \cdot \arctan$ and constant weights. Let $\kappa_{pH}$ be the pH time series. Then, $\kappa_{pH}(t - \Delta t)$, $\kappa_{pH}(t - 2\Delta t)$, $\kappa_{pH}(t - 3\Delta t)$, $\kappa_{pH}(t - 4\Delta t)$ as well as $F_2(t - \Delta t)$, $F_2(t - 2\Delta t)$, $F_2(t - 3\Delta t)$, $F_2(t - 4\Delta t)$ served as inputs for the prediction of $\kappa_{pH}(t)$, where $\Delta t = 0.2$ holds. Again, the classical performance index and the first-order term 3.18 for smoothing the network mapping were used for different network trainings. The classical error criterion results in a mean squared error of 0.030, which equals the value in Sridhar et al. (1998). The constraint training by means of equation 3.18 was able to improve the generalization with a mean squared error of 0.028, which is a slightly better result than the reference value from Sridhar, Bartlett, & Seagrave, (1998).
5 Concluding Remarks

In this work, a new learning algorithm for GDNNs has been presented and tested on nontrivial classification and time-series prediction tasks. The learning procedure was formulated as an optimal control problem with state-dependent constraints. The convergence of the proposed approximation scheme to an optimal solution was also discussed. Galicki et al. (1999) have shown that GDNNs possess many advantages as compared with classical neural networks (constant weights). Nevertheless, the problem of improving the generalization capability by means of mapping curvature for a dynamic network has not been addressed previously, although it is crucial in the normal functioning (i.e., after the learning) of the neural network. The simulation results have unambiguously shown that the application of mapping curvature requires a significantly smaller training set as compared to the sensitivity measure to achieve a good generalization. The generality of the presented approach carries with it a penalty: a somewhat larger number of computations is required to train a GDNN subject to performance index 2.4 or 2.9 (a linear programming problem has to be solved in each iteration in the case of nontrivial state constraint 2.12). However, in our opinion, the capability of good generalization seems to be the most important when applying GDNNs to many practical dynamic tasks. Finally, this work is a first attempt in constructing such algorithms by using the dynamic network curvature.

6 Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, WI 1166/4-1, Ga 652/1-1).

References


Received November 26, 2002; accepted November 3, 2003.