PERGAMON

Neural Networks 14 (2001) 1075–1088

Neural Networks

www.elsevier.com/locate/neunet

Contributed article

A dynamical model for the analysis and acceleration of learning in feedforward networks

Nikolaos Ampazis[a], Stavros J. Perantonis[a], John G. Taylor[b,*]

[a]*Institute of Informatics and Telecommunications, National Center for Scientific Research 'Demokritos', 153 10 Agia Paraskevi, Athens, Greece*
[b]*Department of Mathematics, King's College London, Strand, London WC2R 2LS, UK*

Received 30 November 1998; accepted 16 March 2001

Abstract

A dynamical system model is derived for feedforward neural networks with one layer of hidden nodes. The model is valid in the vicinity of flat minima of the cost function that rise due to the formation of clusters of redundant hidden nodes with nearly identical outputs. The derivation is carried out for networks with an arbitrary number of hidden and output nodes and is, therefore, a generalization of previous work valid for networks with only two hidden nodes and one output node. The Jacobian matrix of the system is obtained, whose eigenvalues characterize the evolution of learning. Flat minima correspond to critical points of the phase plane trajectories and the bifurcation of the eigenvalues signifies their abandonment. Following the derivation of the dynamical model, we show that identification of the hidden nodes clusters using unsupervised learning techniques enables the application of a constrained application (Dynamically Constrained Back Propagation—DCBP) whose purpose is to facilitate prompt bifurcation of the eigenvalues of the Jacobian matrix and, thus, accelerate learning. DCBP is applied to standard benchmark tasks either autonomously or as an aid to other standard learning algorithms in the vicinity of flat minima. Its application leads to significant reduction in the number of required epochs for convergence. © 2001 Published by Elsevier Science Ltd.

Keywords: Multilayer neural networks; Supervised learning; Flat minima; Dynamical systems; Jacobian matrix; Eigenvalues; Constrained optimization

1. Introduction

Multilayer feedforward neural networks have been the preferred neural network architectures for the solution of classification and function approximation problems due to their interesting learning and generalization abilities. From the numerous methods that have been proposed for training multilayered feedforward networks, some, including classic back-propagation, have relatively low complexity per epoch, but are rather inefficient in dealing with extended plateaus (or flat minima) of the cost functions. Other methods are more efficient in dealing with complex topological features of the cost function landscape at the expense of added computational complexity. Notable examples include both off-line and on-line learning paradigms. For example, second order methods related to efficient off-line learning require the evaluation and inversion of the Hessian matrix, which is clearly a computationally very demanding task when the number of parameters is large. The same problem is also eminent in efficient on-line techniques such as the natural gradient descent method (Amari, 1998) which requires the inversion of the Fisher information matrix, for which the computational cost is very large for large-scale problems.

In earlier work (Ampazis, Perantonis & Taylor, 1999a), we have approached the problem of flat minima using a method that originates from the theory of dynamical systems. Motivated by the connection between flat minima and the build up of redundancy, we introduced suitable state variables formed by appropriate linear combinations of the synaptic weights, and we derived a linear dynamical system model for a network with two hidden nodes and a single output. Using that model, we were able to describe the dynamics of such a network in the vicinity of flat plateaus, and we showed that the learning behavior can be characterized by the largest eigenvalue of the Jacobian matrix corresponding to the linearized system. It was shown that in the vicinity of flat minima, learning evolves slowly because this eigenvalue is very small and that the network is able to abandon the minimum only when the eigenvalues of its Jacobian matrix bifurcate.

The study of the nature of flat minima apart from its

* Corresponding author. Tel.: +44-20-7848-2214; fax: +44-20-7848-2017.

E-mail address: udah057@kcl.ac.uk (J.G. Taylor).

0893-6080/01/$ - see front matter © 2001 Published by Elsevier Science Ltd.
PII: S0893-6080(01)00052-1

intrinsic value in advancing research into the dynamics of learning, can have significant impact in the development of new learning methods inspired by deeper understanding of the fundamental mechanisms involved in the dynamical behavior of layered networks. We envisage two types of benefits coming from this approach.

1. Having identified a flat minimum, one can apply a computationally intensive algorithm just for a few epochs until the flat minimum is abandoned, thus reducing the overall computational complexity of the learning process.
2. The insight gained by the analysis of the nature of the flat minima is valuable for proposing tailor-made efficient algorithms for promptly abandoning temporary minima, whose complexity is much lower than related general purpose algorithms. Thus, even for the few epochs that will be needed to abandon the flat minimum there will be a gain in computational cost.

It is our belief that the last statement is true for both on-line and off-line learning. In our earlier work, we concentrated on the off-line mode of learning in order to propose one such tailor-made algorithm. We derived an analytical expression representing an approximation to the largest eigenvalue and introducing an efficient constrained optimizational algorithm that achieves simultaneous minimization of the cost function and maximization of the largest eigenvalue of the Jacobian matrix of the dynamical system model so that the network avoids getting trapped at a flat minimum. As a result, significant acceleration of learning in the vicinity of flat minima was achieved, reducing the total training time. The algorithm was also benchmarked against back-propagation and other well-known variants thereof in classification problems, exhibiting a very good overall behavior.

The purpose of this paper is to extend the dynamical analysis in order to account for a more general type of feedforward networks. We still consider networks with one hidden layer, but place no restriction whatsoever on the number of input, hidden and output nodes. Our study shows that the introduction of suitable state variables results in significant decouplings in the essential quantities related to learning, and, for off-line learning, leads to the formulation of a linear dynamical system model for this more general type of network. In particular, for each cluster of redundant hidden nodes, a linearized system in the corresponding dynamical variables is introduced, which is described by a corresponding symmetric Jacobian matrix with lower dimension than the total number of the weights and thresholds of the network. Abandonment of flat minima arising from the build up of redundancy is signified by the bifurcation of the eigenvalues of the Jacobian matrix of each cluster of redundant hidden units.

Moreover, we extend our effort to incorporate the dynamical system formalism into a learning algorithm that allows successful negotiation of the flat minima and, therefore, accelerates learning. It turns out that such a task requires the ability to identify clusters of redundant hidden nodes, which can be achieved using unsupervised clustering techniques. The identification of individual clusters allows the calculation of the Jacobian eigenvalues of the dynamical system model and the application of extended constrained learning optimization techniques that enable prompt bifurcation of the eigenvalues. A training algorithm (Dynamically Constrained Back Propagation—DCBP) ensues, which can be applied either autonomously or as an aid, in the vicinity of flat minima, to other well-known supervised learning algorithms. In the experimental section it is shown that DCBP exhibits improved learning abilities compared to standard back-propagation and to other reputedly fast learning algorithms (resilient propagation, ALECO-2 and variations of the conjugate gradient methods) in standard benchmark tasks.

The paper is organized as follows: in Section 2 we introduce the dynamical variables for arbitrary networks with a single hidden layer and we discuss the relation of the corresponding dynamical system model arising in the off-line learning mode to other on-line techniques dealing with the flat minima problem. In Section 3 we introduce the constrained optimization method designed to facilitate learning using constraints imposed on the eigenvalues of the Jacobian matrix. In Section 4 we present an outline of the steps required by the proposed DCBP algorithm. Section 5 contains our simulation results and describes the experiments conducted to test the performance of the algorithm and compare it with that of other supervised learning algorithms, Finally, in Section 6 conclusions are drawn and future work is outlined.

2. The dynamical analysis

2.1. Motivation

Consider a neural network with a single hidden layer which has N external input signals with the addition of a bias input. The bias signal is identical for all neurons in the network. The hidden layer consists of M neurons and the output layer contains K neurons with sigmoid activation functions $f(s = 1/(1 + \exp(-s))$. For a given training pattern p, the square error cost function is

$$E_p = \frac{1}{2}\sum_{i=1}^{K}(d_i - y_i)^2 \tag{1}$$

where y_i denote the output activations and d_i are the desired responses of each output node i. The gradient components of the cost function of Eq. (1) corresponding to the hidden-to-output connections are given by:

$$\frac{\partial E_p}{\partial w_{ij}} = (d_i - y_i)y_i(1 - y_i)y_j \tag{2}$$

where w_{ij} are the weight connections between each hidden

node j and output node i, and y_j are the outputs of each hidden unit (with $j = 0$ corresponding to the bias signal).

The gradient components corresponding to the input-to-hidden connections are:

$$\frac{\partial E_p}{\partial w_{jk}} = \sum_i (d_i - y_i) y_i (1 - y_i) w_{ij} y_j (1 - y_j) x_k \tag{3}$$

where w_{jk} represents the weight connection between hidden node j and input node k, and x_k is the signal from input node k.

In order to maintain a consistent terminology, for the rest of the paper we will always make a reference to output nodes using the subscript i, to hidden nodes using the subscript j, and finally to input nodes using the subscript k.

Written in vector notation, for each hidden node j, Eq. (3) becomes

$$\frac{\partial E_p}{\partial \mathbf{w}_j} = \sum_i (d_i - y_i) y_i (1 - y_i) w_{ij} y_j (1 - y_j) \mathbf{x} \tag{4}$$

where

$$\mathbf{w}_j = (w_{j0} \cdots w_{jN})^T \tag{5}$$

For the network described above, consider a number of M_c hidden nodes that form a particular cluster C which contains hidden units with similar activations. Motivated by our earlier work (Ampazis et al., 1999a) which indicates that for any given data set flat minima occur because of the building of redundancy, we expect that appropriate dynamical variables can be selected using the weights connected to hidden nodes j belonging to C. In particular, we can consider the following dynamical variables:

- The differences of each hidden weight vector $\mathbf{w}_j$ from the average of all $\mathbf{w}_j (j \in C)$.
- The differences of each weight component w_{ij} of a single output unit from the average of all weights $w_{ij} (j \in C)$.

Hence, we define

$$\boldsymbol{\omega}^c = \frac{\sum_{j \in C} \mathbf{w} j}{M_c} \tag{6}$$

and

$$\boldsymbol{\epsilon}_j = \mathbf{w}_j - \boldsymbol{\omega}^c, \qquad j \in C \tag{7}$$

Similarly for each output node i, we define

$$v_i^c = \frac{\sum_{j \in C} w_{ij}}{M_c} \tag{8}$$

and

$$\mu_{ij} = w_{ij} - v_i^c, \qquad j \in C \tag{9}$$

Taking into account Eqs. (2) and (7), the above equation implies that

$$\frac{\partial E_p}{\partial \mu_{ij}} = (d_i - y_i) y_i (1 - y_i) \times \left\{ f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j) \cdot \mathbf{x}] - \frac{1}{M_c} \sum_{J \in C} f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_J) \cdot \mathbf{x}] \right\} \tag{10}$$

Similarly, taking into account Eqs. (4) and (9), Eq. (7) implies that

$$\frac{\partial E_p}{\partial \boldsymbol{\epsilon}_j} = \sum_i (d_i - y_i) y_i (1 - y_i) \mathbf{x} \left\{ f'[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j) \cdot \mathbf{x}](\mu_{ij} + v_i^c) - \frac{1}{M_c} \sum_{J \in C} f'[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j) \cdot \mathbf{x}](\mu_{iJ} + v_i^c) \right\} \tag{11}$$

2.2. The dynamical system model

The introduction of the quantities $\boldsymbol{\epsilon}_j$ and μ_{ij} allows for the description of the network by a set of reduced variables which takes into account the underlying redundancy exhibited by the original variables, namely all the weights and thresholds of the multilayer network. There are multiple benefits arising from such a description. These benefits are closely related to the way by which we wish to proceed with our analysis of the learning process. The first line of approach is to consider the batch mode of training which is evidently more suitable than on-line learning for the derivation of a deterministic dynamical system model. Given a small learning rate η, the difference update equations of the variables $\boldsymbol{\epsilon}_j$ and μ_{ij} of the batch gradient descent algorithm can be approximated by *differential equations* in time. Therefore, for batch learning, Eqs. (10) and (11) yield

$$\dot{\mu}_{ij} = \sum_p (d_i^{(p)} - y_i^{(p)}) y_i^{(p)} (1 - y_i^{(p)}) \times \left[f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j) \cdot \mathbf{x}^{(p)}] - \frac{1}{M_c} \sum_{J \in C} f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_J) \cdot \mathbf{x}^{(p)}] \right] \tag{12}$$

and

$$\dot{\boldsymbol{\epsilon}}_j = \sum_{pi} (d_i^{(p)} - y_i^{(p)}) y_i^{(p)} (1 - y_i^{(p)}) \mathbf{x}^{(p)} \left[f'[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j) \cdot \mathbf{x}^{(p)}](\mu_{ij} + v_i^c) - \frac{1}{M_c} \sum_{J \in C} f'[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_J) \cdot \mathbf{x}^{(p)}](\mu_{iJ} + v_i^c) \right] \tag{13}$$

respectively.

In the above equations, the summation is carried out over all input patterns, $d_i^{(p)}$ represents the desired response for output node i at the presentation of pattern p, $y_i^{(p)}$ is the activation of output node i at the presentation of that pattern, and $\mathbf{x}_i^{(p)}$ represents the p-th input pattern vector.

We can immediately observe that the above equations represent a set of dynamical system equations of the form

$$\dot{\boldsymbol{\epsilon}}_j = \mathbf{F}(\boldsymbol{\epsilon}_J, \mu_{ij}), \qquad \dot{\mu}_{ij} = G(\boldsymbol{\epsilon}_J, \mu_{iJ}) \tag{14}$$

It is clear that if we set all $\boldsymbol{\epsilon}_J = 0$ and $\mu_{iJ} = 0$, then $\mathbf{F}(\boldsymbol{\epsilon}_J, \mu_{iJ}) = 0$ and $G(\boldsymbol{\epsilon}_J, \mu_{iJ}) = 0$. Therefore, the judicious choice of state variables has resulted in the mapping of the flat minimum to this origin, which is a critical point of the system. Moreover, $\mathbf{F}(\boldsymbol{\epsilon}_J, \mu_{iJ})$ and $G(\boldsymbol{\epsilon}_J, \mu_{iJ})$ are twice-differentiable and we can, therefore, proceed with the linearization of the system.

Using Taylor's theorem for the approximation of f near the point $\boldsymbol{\omega}^c \cdot \mathbf{x}^{(p)}$, and keeping only first order terms, the linear system of differential equations that describes the dynamics of the network in terms of the state variables $\boldsymbol{\epsilon}_j$ and μ_{ij} can be obtained as follows:

First, consider the factors in brackets appearing on the right-hand-side of Eqs. (12) and (13):

$$B_{jc}^{(p)} = f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j)\cdot\mathbf{x}^{(p)}] - \frac{1}{M_c}\sum_{J\in C} f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_J)\cdot\mathbf{x}^{(p)}] \tag{15}$$

and

$$C_{ijc}^{(p)} = f'[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_j)\cdot\mathbf{x}^{(p)}](\mu_{ij} + v_i^c) - \frac{1}{M_c}\sum_{J\in C} f'[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_J)\cdot\mathbf{x}^{(p)}](\mu_{iJ} + v_i^c) \tag{16}$$

Evidently, if we set all $\boldsymbol{\epsilon}_j$ and μ_{ij} equal to zero, we get $B_{jc}^{(p)} = 0$ and $C_{ijc}^{(p)} = 0$, so that $B_{jc}^{(p)}$ and $C_{ijc}^{(p)}$ contain no zero-th order (constant) term. It follows that to expand the right-hand-sides of Eqs. (12) and (13) up to first order, we need only consider the expansion of $y_j^{(p)}$ and its derivative up to first order, as well as the zero-th order of $y_i^{(p)}$. Therefore, we can set:

$$y_j^{(p)} f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) + f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})(\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)}) \tag{17}$$

and

$$f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) = f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})[1 - 2f\left(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}\right)](\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)}) + f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) \tag{18}$$

Moreover, since

$$y_i^{(p)} = f\left[\sum_{J\in C}(v_i^c + \mu_{iJ})f[(\boldsymbol{\omega}^c + \boldsymbol{\epsilon}_J)\cdot\mathbf{x}^{(p)}] + \sum_{J\notin C} w_{iJ}y_J^{(p)}\right] \tag{19}$$

the zero-th order term of $y_i^{(p)}$ will be given by:

$$y_i^{0(p)} = f\left[M_c v_i^c f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) + \sum_{J\notin C} w_{iJ}y_J^{(p)}\right] \tag{20}$$

It follows from Eqs. (6) and (7) that

$$\sum_{j\in C}\boldsymbol{\epsilon}_j = 0 \tag{21}$$

Similarly, it follows from Eqs. (8) and (9) that

$$\sum_{j\in C}\mu_{ij} = 0 \tag{22}$$

We are now ready to proceed with the evaluation of $\dot{\mu}_{ij}$. Evidently, to first order, we have:

$$B_{jb}^{(p)} = f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) + f'\left(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}\right)(\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)}) - \frac{1}{M_c}\sum_{J\in C} f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) - \frac{1}{M_c}\sum_{J\in C} f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)}) \tag{23}$$

The first and third terms on the right-hand-side of Eq. (23) cancel each other and the fourth term is equal to zero on account of Eq. (21). It follows that, up to first order:

$$B_{jc}^{(p)} = f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})(\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)}) \tag{24}$$

from which we can readily obtain the following equation for $\dot{\mu}_{ij}$:

$$\dot{\mu}_{ij} = \sum_p (d_i^{(p)} - y_i^{0(p)})y_i^{0(p)}(1 - y_i^{0(p)})f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})(\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)}) \tag{25}$$

Similarly, in order to evaluate $\boldsymbol{\epsilon}_j$, we get, up to first order:

$$\begin{aligned} C_{ijc}^{(p)} = {} & f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})v_i^c + f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})\mu_{ij} \\ & + [1 - 2f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})]f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})v_i^c(\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)}) \\ & - \frac{1}{M_c}\sum_{J\in C} f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})v_i^c - \frac{1}{M_c}\sum_{J\in C} f'(\boldsymbol{\omega}^c\cdot\mathbf{x})\mu_{iJ} \\ & - \frac{1}{M_c}\sum_{J\in C}[1 - 2f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})]f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})v_i^c(\boldsymbol{\epsilon}_J\cdot\mathbf{x}^{(p)}) \end{aligned} \tag{26}$$

On the right-hand-side of Eq. (26), the first and fourth terms cancel each other, while the fifth and sixth terms vanish on account of Eqs. (22) and (21). It follows that:

$$C_{ijc}^{(p)} = f'(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})[\mu_{ij} + v_i^c[1 - 2f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})](\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)})] \tag{27}$$

from which the following equation for $\boldsymbol{\epsilon}_j$ is readily obtained:

$$\dot{\boldsymbol{\epsilon}}_j = \sum_{pi}\left(d_i^{(p)} - y_i^{0(p)}\right)y_i^{0(p)}\left(1 - y_i^{0(p)}\right)f'\left(\omega^c\mathbf{x}^{(p)}\right)\mathbf{x}^{(p)} \times\{\mu_{ij} + v_i[1 - 2f(\boldsymbol{\omega}^c\cdot\mathbf{x}^{(p)})](\boldsymbol{\epsilon}_j\cdot\mathbf{x}^{(p)})\} \tag{28}$$

Eqs. (28) and (25) are the fundamental differential equations describing the dynamics of the back-propagation system in terms of the state variables $\boldsymbol{\epsilon}_j$ and μ_{ij}. From these equations we observe that time derivatives of state variables corresponding to a certain hidden node j depend only state variables corresponding to the same hidden node. Therefore, introducing the vector $\mathbf{u}_j = (\boldsymbol{\epsilon}_j, \mu_{ij})^T$, $i = 1, \ldots, K$ we obtain M_c equations of the form:

$$\dot{\mathbf{u}}_j = \mathbf{J}_c\mathbf{u}_j \tag{29}$$

where the Jacobian matrix $\mathbf{J}_c$ is given by

$$\mathbf{J}_c = \sum_p f'(\boldsymbol{\omega}^c \cdot \mathbf{x}^{(p)}) \times \begin{pmatrix} \sum_i A_i^{(p)}[1 - 2f(\boldsymbol{\omega}^c \cdot \mathbf{x}^{(p)})]\mathbf{x}^{(p)}\mathbf{x}^{(p)T} v^c & \mathbf{A}^{(p)}\mathbf{x}^{(p)} \\ (\mathbf{A}^{(p)}\mathbf{x}^{(p)})^T & 0 \end{pmatrix} \tag{30}$$

with

$$\mathbf{A}^{(p)} = ((d_i^{(p)} - y_i^{0(p)})y_i^{0(p)}(1 - y_i^{0(p)}), \qquad i = 1, \ldots, K) \tag{31}$$

Eq. (30) constitutes a generalization of the expression that we had obtained in our earlier work (Ampazis et al., 1999a) for the Jacobian matrix of a network with two hidden nodes and a single output unit. Note that the dimension of all vectors $\mathbf{u}_j$ is $Q = N + K + 1$ and, therefore, all Jacobian matrices are of dimension $Q \times Q$ (independently of the cluster or specific hidden node to which the corresponding dynamical variables are associated). Moreover, all Jacobian matrices associated with a certain cluster are equal, so that in effect we have one representative Jacobian matrix for each cluster. All Jacobian matrices are real and symmetric and, therefore, all their eigenvalues are real. In addition, all the corresponding eigenvectors are linearly independent and if there are no eigenvalue multiplicities they will form an orthogonal set. Since the eigenvalues are real, this means that flat minima correspond to stationary points and not to spiral or center points of the phase plane. As a consequence, small perturbations in the eigenvalues will not affect the stability or instability of the system.

Due to the exponential nature of the solutions, the followed trajectory will depend on the magnitude of the largest eigenvalue for each Jacobian matrix. Initially, all eigenvalues are small in magnitude and the network spends a relatively long time in the vicinity of the critical point. As time passes, since all the eigenvalues do not differ too much, small perturbations due to the continuous update of the weights at each epoch cause them eventually to bifurcate, so that the system is able to follow a trajectory which allows it to move far away from the critical point. Asymptotically, the trajectories followed by the dynamical systems described by Eq. (29) are parallel to the eigenvectors corresponding to the maximum Jacobian eigenvalues. Thus, after the bifurcation of the eigenvalues has occurred, weight updates approximately follow the rule:

$$d\mathbf{u}_j \approx \text{sign}(\mathbf{u}_j^T \boldsymbol{\xi}^c)\boldsymbol{\xi}^c \tag{32}$$

where $\boldsymbol{\xi}^c$ is the eigenvector of $\mathbf{J}_c$ corresponding to the maximum eigenvalue λ_c.

2.3. Relation to other methods

A second line of approach that can take into account the benefits arising from the description of the network by the set of reduced variables is to consider the on-line mode of training. With such an approach it is possible to describe the learning task using concepts from the field of information geometry (Amari, 1985; Amari & Nagaoka, 2000). According to such a description the parameter space of network variables has a Riemannian structure (Amari, 1998; Yang & Amari, 1998) which can be applied for the definition of the direction of steepest decrease of the cost function given by Eq. (1). For *stochastic* multilayer perceptrons, the Riemannian metric tensor $G(\mathbf{W})$ is given by the Fisher information matrix (Amari, 1998) with elements

$$g_{ij}(\mathbf{W}) = E\left[\frac{\partial \tilde{E}_p}{\partial W_i}\frac{\partial \tilde{E}_p}{\partial W_j}\right] \tag{33}$$

In the above expression, $\mathbf{W}$ is the column vector containing all the weights and thresholds of the network, $E[.]$ denotes the expectation with respect to the input–output mapping for the entire training set and $\tilde{E}_p$ is the log likelihood given by

$$\tilde{E}_p = \sum_i \left\{ -\frac{1}{2\sigma}(d_i - < y_i >)^2 - \log\left(\sqrt{2\pi\sigma}\right)\right\} \tag{34}$$

which is the logarithm of the set of all conditional probability distributions (with variance σ) of desired outputs d_i conditioned on input $\mathbf{x}$.

The steepest descent direction in the Riemannian parameter space is the direction of the natural gradient of the cost function given by

$$\tilde{\nabla} E_p = \mathbf{G}^{-1} \nabla E_p \tag{35}$$

This suggests the *natural gradient descent* algorithm of the form

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t \tilde{\nabla} E_p \tag{36}$$

where η_t represents the learning rate which may depend on t. It has been proved (Amari, 1998) that the on-line learning method based on the natural gradient is asymptotically as efficient as the optimal batch algorithm, while it has been suggested (Yang & Amari, 1998) that the natural gradient algorithm may avoid or alleviate the flat plateau phenomena corresponding to temporary minima. Similar ideas are also explored in a recently published paper (Fukumizu & Amari, 2000) which examines the behavior of hierarchical structures of multilayer perceptrons. However, there are some important problems related to the implementation of the natural gradient descent algorithm. Firstly, the explicit form of the Fisher information matrix requires knowledge of the input distribution which is usually unknown. Secondly, even if that distribution is known the algorithm faces the problem of evaluating and inverting the Fisher information matrix, both of which are computationally expensive operations. In order to overcome these problems, an adaptive method for obtaining an estimate of the natural gradient has recently been proposed (Amari, Park & Fukumizu, 1999) that does not require any information on the input distribution and inversion of the explicit form of

the Fisher information matrix. However, a potential problem of the method is that the updates of the estimates of the matrix might not converge to the true Fisher information matrix because the algorithm is extremely sensitive to the proper selection of two learning rate parameters which are currently selected heuristically (Amari et al., 1999).

Based on the information geometry approach for the stochastic multilayer perceptron, it is possible to take into account the description of the network by the set of reduced variables derived earlier in order to deal with the implementation problems of the natural gradient algorithm. Its first problem can be partially overcome by noting that the cost function of Eq. (34) can be regarded as the negative of the square of an error when d_i is a desired target value given $\mathbf{x}$, except for a scale and a constant term (Amari et al., 1999). Hence, the maximization of the likelihood can be regarded as equivalent to the minimization of the cost function of Eq. (1). Taking this fact into account, its second problem can be substantially reduced if we consider the stochastic multilayer perceptron consisting of M_c redundant hidden nodes. This multilayer perceptron can be described with the reduced set of variables and therefore the elements of its Fisher information matrix are given by

$$g_{ij}(\boldsymbol{\theta}^j) = E\left[\frac{\partial E_p}{\partial \theta_i^j}\frac{\partial E_p}{\partial \theta_j^i}\right] \tag{37}$$

where $\boldsymbol{\theta}^j = (\boldsymbol{\epsilon}_j^T, \mu_{ij})^T$. The above expression represents a significant reduction in the total number of variables involved in the explicit calculation of the Fisher information matrix which is independent (to first order) from the set of variables themselves (all its elements can be evaluated by Eqs. (25) and (28), dropping, of course, the pattern summation). Hence, as regards these parameters, the space is flat (although not necessarily Cartesian). It would be interesting to investigate the implications of this fact on a suitably modified cost-effective implementation of a natural gradient descent algorithm in the vicinity of flat minima in order to accelerate learning. To this end, preliminary studies have already been made by the authors and their results will be presented elsewhere.

3. Constrained optimization method

In this section, we concentrate on the utilization of the information provided by the dynamical system model for off-line learning in order to explore potential ways of helping the network to escape from flat minima. Following the analysis of the previous section, it is evident that if the maximum eigenvalues $\lambda_c, c = 1, \ldots, S$ of the Jacobian matrices $\mathbf{J}_c$ of Eq. (30) corresponding to each of the S clusters of hidden nodes are relatively large, then the network is able to escape from the flat minimum. Hence, instead of waiting for the growth of the eigenvalues, the objective of our new approach is to raise these eigenvalues more rapidly using an appropriate constrained optimization method. Since it is difficult in the general case to express the maximum eigenvalues in closed form in terms of the weights, we choose to raise the values of appropriate lower bounds $\boldsymbol{\Phi}_c \leq \lambda_c$ for these eigenvalues, which are obtained as follows. It is well known from linear algebra that since $\mathbf{J}_c$ is a real and symmetric matrix, then

$$\mathbf{z}^T\mathbf{J}_c\mathbf{z} \leq \lambda_c \mathbf{z}^T\mathbf{z} \forall \mathbf{z} \in R^Q \tag{38}$$

Denoting by $\mathbf{1_Q}$ the vector with Q elements which are all equal to 1, for the constrained optimization method we use

$$\boldsymbol{\Phi}_c = \frac{1}{Q}(\mathbf{1}_\mathbf{Q}^T\mathbf{J}_c\mathbf{1}_\mathbf{Q}) \tag{39}$$

which means that the product in the left-hand-side of Eq. (38) is simply the sum of the elements of the matrix. Thus, we are able to obtain an analytic expression for a lower bound of the maximum eigenvalue by directly evaluating the sum of the elements of $\mathbf{J}_c$ as given by Eq. (30). Note that according to Eqs. (30) and (39), each $\boldsymbol{\Phi}_c$ depends on the corresponding values of the parameters $\boldsymbol{\omega}^c$ and $\mathbf{v}^c = (v_i^c, i = 1, \ldots, K)^T$ for all identified clusters $c = 1, \ldots, S$. Hence, the optimization should be attempted with respect to these parameters only, whose total number is $S\cdot(N + K + 1)$. Since $S \leq M$, this is always smaller than the total number of free network parameters (biases and weights) which is equal to $M\cdot(N + K) + M + K$, and this reduces significantly the computational complexity.

For this rest of this section, it is convenient, for reasons of compactness in terminology to

- group all weight vectors connected to a certain hidden node j into a single column vector $\mathbf{W}_i = (\mathbf{w}_j^T, w_{ij})^T$, $i = 1, \ldots, K$;
- construct for each cluster c a vector containing all the appropriate weight averages: $\boldsymbol{\Omega}_c(\boldsymbol{\omega}^{cT}, \mathbf{v}^{cT})^T$; and
- group all $\boldsymbol{\Omega}_c$ corresponding to all clusters into a vector $\boldsymbol{\Omega} = (\boldsymbol{\Omega}_c^T, c = 1, \ldots, S)^T$.

To achieve rapid growing of the maximum eigenvalues, we propose an iterative algorithm for adapting $\boldsymbol{\Omega}$, whose objectives are outlined as follows.

1. At each epoch of the learning process, the vector $\boldsymbol{\Omega}$ will be incremented by $\mathrm{d}\boldsymbol{\Omega}$, so that the search for an optimum new point in the space of $\boldsymbol{\Omega}$ is restricted to a hypersphere of known radius δP centered at the point defined by the current $\boldsymbol{\Omega}$. If δP is small enough, the changes to the cost function E and to $\boldsymbol{\Phi}_c$ induced by changes in the weights can be approximated by the first differentials $\mathrm{d}E$ and $\mathrm{d}\boldsymbol{\Phi}_c$.
2. At each epoch it is desirable to raise each $\boldsymbol{\Phi}_c$ as much as possible. Note that this corresponds to a multiobjective optimization problem, to which various alternative schemes can be applied. We have chosen to raise each $\boldsymbol{\Phi}_c$ at a constant rate κ, i.e. $\mathrm{d}\boldsymbol{\Phi}_c = \kappa$. In this expression, the maximum possible value of κ should be used. We mention, in passing, that other schemes were also tried,

but with limited success. For example, exponential rates of change of the form $d\mathbf{\Phi}_c = \kappa\mathbf{\Phi}_c$ were found to lead to premature growth of the weights in their efforts to raise $\mathbf{\Phi}_c$.

3. Maximization of κ must be attempted subject to the constraint that the state of the network should remain in the vicinity of the stationary point, so that our dynamical system analysis remains valid. We can ensure this by insisting that there is no change in the value of the cost function E so that the relation $dE = 0$ should hold at each epoch of the proposed algorithm.

Based on these considerations, the following optimization problem can be formulated for each epoch of the algorithm, whose solution will determine the adaptation rule for $\mathbf{\Omega}$:

Maximize κ with respect to $d\mathbf{\Omega}$, subject to the following constraints:

$$d\mathbf{\Omega}^T d\mathbf{\Omega} = (\delta P)^2 \tag{40}$$

$$\kappa = d\mathbf{\Phi}_c, \qquad c = 1, \ldots, S \tag{41}$$

$$dE = 0 \tag{42}$$

This constrained optimization problem can be solved analytically by introducing two Lagrange multipliers L_1 and L_2 to take account of Eqs. (42) and (40), respectively, and a vector of multipliers $\mathbf{\Lambda}$ to take account of Eq. (41). We thus introduce the quantity

$$\kappa' = \kappa + L_1 dE + L_2[d\mathbf{\Omega}^T d\mathbf{\Omega} - (\delta P)^2] + \sum_{c=1}^{S} \mathbf{\Lambda}_c(d\mathbf{\Phi}_c - \kappa) \tag{43}$$

On evaluating the differentials involved in the right-hand-side, we readily obtain:

$$\kappa' = \kappa + L_1(\mathbf{G}^T d\mathbf{\Omega}) + L_2[d\mathbf{\Omega}^T d\mathbf{\Omega} - (\delta P)^2] + \mathbf{\Lambda}^T(\mathbf{F}d\Omega - \kappa\mathbf{1_S}) \tag{44}$$

where $\mathbf{G}$ is a vector and $\mathbf{F}$ is a matrix whose elements are given by

$$\mathbf{G}_j = \frac{\partial E}{\partial \Omega_j}, \qquad \mathbf{F}_{cj} = \frac{\partial \Phi_c}{\partial \Omega_j} \tag{45}$$

To maximize κ under the required constraints, we demand that:

$$d\kappa' = d\kappa(1 - \mathbf{\Lambda}^T\mathbf{1_S}) + (L_1\mathbf{G}^T + \mathbf{\Lambda}^T\mathbf{F} + 2L_2 d\mathbf{\Omega}^T)d^2\mathbf{\Omega} = 0 \tag{46}$$

$$d^2\kappa' = 2L_2 d^2\mathbf{\Omega}^T d^2\mathbf{\Omega} < 0 \tag{47}$$

Hence, the factors multiplying $d^2\mathbf{\Omega}$ and $d\kappa$ in Eq. (46) should vanish, and, therefore, we obtain:

$$d\mathbf{\Omega} = -\frac{L_1}{2L_2}\mathbf{G} - \frac{1}{2L_2}\mathbf{F}^T\mathbf{\Lambda} \tag{48}$$

$$\mathbf{\Lambda}^T\mathbf{1_S} = 1 \tag{49}$$

By left multiplication of both sides of Eq. (48) by $\mathbf{G}^T$ and by taking into account Eq. (42) we obtain a relation between L_1 and $\mathbf{\Lambda}$. Solving for L_1 yields

$$L_1 = -\frac{1}{I_{GG}}\mathbf{\Lambda}^T\mathbf{I_{GF}} \tag{50}$$

where

$$I_{GG} = \mathbf{G}^T\mathbf{G}, \qquad \mathbf{I_{GF}} = \mathbf{FG} \tag{51}$$

Substituting Eq. (50) into Eq. (48) we arrive at the following equation which constitutes the weight update rule for the neural network

$$d\mathbf{\Omega} = \frac{1}{2L_2}\left(\frac{\mathbf{\Lambda}^T\mathbf{I_{GF}}}{I_{GG}}\mathbf{G} - \mathbf{F}^T\mathbf{\Lambda}\right) \tag{52}$$

provided that L_2 and $\mathbf{\Lambda}$ can be evaluated in terms of known quantities. To carry out these evaluations we proceed as follows. By left multiplication of both sides of Eq. (52) by $\mathbf{F}$ and taking into account Eq. (41), we obtain

$$\kappa\mathbf{1_S} = -\frac{\mathbf{R\Lambda}}{2L_2} \tag{53}$$

where the matrix $\mathbf{R}$ is defined by:

$$\mathbf{R} = \frac{1}{I_{GG}}(I_{GG}\mathbf{I_{FF}} - \mathbf{I_{GF}} \times \mathbf{I_{GF}}) \tag{54}$$

with

$$\mathbf{I_{FF}} = \mathbf{F}^T\mathbf{F} \tag{55}$$

Solving Eq. (53) for $\mathbf{\Lambda}$ yields

$$\mathbf{\Lambda} = -2L_2\kappa\mathbf{R}^{-1}\mathbf{1_S} \tag{56}$$

Upon substitution of this equation into Eq. (49) we find that

$$2L_2\kappa = -\frac{1}{(\mathbf{R}^{-1})_a} \tag{57}$$

where $(.)_a$ denotes the sum of all elements of a matrix.

Eqs. (56) and (57) lead to the evaluation of $\mathbf{\Lambda}$ as follows:

$$\mathbf{\Lambda} = \frac{\mathbf{R}^{-1}\mathbf{1_S}}{(\mathbf{R}^{-1})_a} \tag{58}$$

It remains to calculate L_2. To this end, we substitute Eq. (52) into Eq. (40).

After some algebra, we arrive at the following result:

$$4L_2^2 I_{GG}(\delta P)^2 = I_{GG}(\mathbf{\Lambda}^T\mathbf{I_{FF}}\mathbf{\Lambda}) - (\mathbf{\Lambda}^T\mathbf{I_{GF}})^2 \tag{59}$$

Using the definition of $\mathbf{R}$ given by Eq. (54) we can rewrite the last equation in a more compact form:

$$4L_2^2(\delta P)^2 = \mathbf{\Lambda}^T\mathbf{R\Lambda} \tag{60}$$

To solve for L_2, we must make sure that the expression on the right-hand-side is positive. Fortunately, the matrix $\mathbf{r}$ is positive definite by definition. To see this, let us define

$\mathbf{Q} = \mathbf{F}^T\mathbf{\Lambda}$. Using Eqs. (51), (54) and (55) we can write

$$\mathbf{\Lambda}^T\mathbf{R}\mathbf{\Lambda} = \frac{1}{\mathbf{G}^T\mathbf{G}}[(\mathbf{G}^T\mathbf{G})(\mathbf{Q}^T\mathbf{Q}) - \|\mathbf{Q}^T\mathbf{G}\|^2] \tag{61}$$

whereupon the positive definiteness of $\mathbf{R}$ follows from Schwartz's inequality. Combining Eqs. (58) and (59) we now readily evaluate L_2 as follows:

$$L_2 = -\frac{1}{2\delta P}\sqrt{\frac{1}{(\mathbf{R}^{-1})_a}} \tag{62}$$

where the negative square root value has been chosen for L_2 in order to satisfy relation (47). Hence, the evaluation of all Lagrange multipliers in terms of known quantities is now complete.

Using the update rule as given by Eq. (52), for each hidden node $j \in C$ the corresponding weights are changed according to

$$\mathrm{d}\mathbf{W}_j = \mathrm{d}\mathbf{\Omega}_c \tag{63}$$

and thus we are able to raise the maximum Jacobian eigenvalues rapidly, so that the system is able to escape from the stationary point. Since the escape direction for each $\mathbf{u}_j$ is asymptotically parallel to the eigenvector $\boldsymbol{\xi}^c$, we can further speed up learning by insisting that after λ_c has been raised sufficiently, the constrained optimization algorithm is terminated and the weights are updated for just epoch using

$$\mathrm{d}\mathbf{W}_j = \delta P \mathrm{sign}\left(\mathbf{u}_i^T\boldsymbol{\xi}^c\right)\boldsymbol{\xi}^c \tag{64}$$

4. DCBP algorithm outline

In order to formulate a training strategy which takes into account both the dynamical system analysis and the constrained optimization method, we should ensure that we are able to identify the clusters that are formed during the training process. For the cluster identification problem, it should be clear that normally it is difficult to obtain a clear sense of how many clusters are formed during training, but one can only suspect their formation when the error improvement is very small (e.g. for the last 50 epochs). Hence, for this task, an unsupervised clustering algorithm should be employed. Unsupervised clustering algorithms are generally slow, but, fortunately, the number of data points to be clustered are few, namely the number of hidden nodes of the network. In this case, the problem of identifying clusters can be split in two phases. The first phase involves the estimation of the number of clusters and the cluster centers, while the second phase concerns the assignment of each hidden node to the cluster to which it is closer. For the first phase, since we do not have a clear idea of how many clusters are present, we can employ the method of subtractive clustering which is an extension of the mountain clustering method (Chiu, 1994; Yager & Filev, 1994). Subtractive clustering is a fast, one-pass algorithm for estimating the number of clusters and the cluster centers in a set of data. For the second phase, we use the fuzzy c-means algorithm which is an extension to the classical c-means algorithm (Bezdek, 1981) in order to handle partial memberships of the data points to several candidate clusters. The fuzzy c-means algorithm requires only one input parameter, namely the number of identified clusters, and, therefore, accepts as input the output of the subtractive clustering method.

Having identified the number of clusters of redundant hidden nodes and assigning each hidden unit to the cluster in which it belongs, we are able to use Eqs. (52) and (63) at the flat minimum in order to help the network to escape. Following the abandonment of the flat minimum signified by the bifurcation of the eigenvalues of each representative Jacobian matrix for each cluster, the training process can be continued using either standard batch back-propagation or a more effective learning algorithm.

With the above motivations in mind, the proposed Dynamically Constrained Back Propagation (DCBP) algorithm therefore utilizes:

- an unsupervised clustering algorithm for the identification of clusters of redundant hidden nodes;
- a composite weight update rule given by Eqs. (52) and (63) at the points where the network is trapped at a flat minimum; and
- standard off-line back-propagation when the stationary point has been abandoned.

In the rest of the section, we utilize the implementation details of the steps required by the algorithm.

1. **Initialization:** Initialize all free parameters of the network (weights and thresholds) to a small range of values $[-q, q]$, (usually $q \leq 0.1$).
2. **Initial constrained weight update:** Due to the initialization with small weights, in the beginning of training all hidden nodes form a single cluster. Hence, calculate the Jacobian matrix as given by Eq. (30) as well as its maximum and minimum eigenvalues using a suitable numerical method. Update all free parameters according to Eqs. (52) and (63).
3. **Initial alignment:** As soon as the absolute difference between the maximum and minimum eigenvalues of the Jacobian matrix becomes T_0 times larger than its initial value (where T_0 is an arbitrarily chosen threshold)—signifying the bifurcation of the eigenvalues—calculate the eigenvector $\boldsymbol{\xi}^c$ corresponding to the maximum eigenvalue. Align all weight updates $\mathrm{d}\mathbf{W}_j$ ($j \in C$), with $\boldsymbol{\xi}^c$ according to Eq. (64) for just one epoch. Steps 2 and 3 are sufficient for the breaking of the initial symmetry.
4. **Back-propagation:** In the epoch immediately succeeding the alignment, continue training using standard back-propagation (preferably with momentum) while the error

Table 1
Experimental results for the 3-bit parity problem. Comparison of DCBP with five standard learning algorithms for feedforward networks

Algorithms	DCBP	BPM	ALECO-2	RPROP	CG/PR	CG/FR
Parameters	$\delta P = 0.2$ $\xi = 0.5$ $\eta = 0.7$ $\alpha = 0.9$ $T_0 = 2.0$	$\eta = 0.7$ $\alpha = 0.9$	$\delta P = 0.6$ $\xi = 0.85$	$\eta^+ = 1.2$ $\eta^- = 0.5$ $\Delta_{max} = 1.0$ $\Delta_{min} = 10^{-6}$ $\Delta_0 = 0.1$		
# epochs	79	310	69	71	26	49
Successes (%)	95.0	97.0	97.0	60.0	88.0	77.0

reduction for a number of epochs is above a certain threshold.

5. **Cluster identification:** Enable the subtractive clustering procedure to identify the number of clusters, followed by the fuzzy c-means algorithm to assign each hidden node to the cluster to which it belongs.
6. **Constrained weight update:** Calculate the Jacobian matrices of each cluster as given by Eq. (30) as well as their maximum and minimum eigenvalues using a suitable numerical method. Update all free parameters according to Eqs. (52) and (63).
7. **Alignment:** As in step 3, calculate the absolute differences between the maximum and minimum eigenvalues of the Jacobian matrices. As soon as the minimum of these differences satisfies the T_0 threshold condition, calculate the eigenvectors $\boldsymbol{\xi}^c$ corresponding to the maximum eigenvalues and align all $d\mathbf{W}_j$ ($j \in C$), with $\boldsymbol{\xi}^c$ according to Eq. (64) for one epoch.
8. **Termination:** Cycle through steps 4–7 until the error goal has been reached or until no further error reduction is possible.

5. Simulation results

In our simulations, we studied the dynamics of feedforward networks that were trained to solve two different parity problems and a real world classification problem from the PROBEN1 database (Prechelt, 1994). In particular we studied the 3-bit and 4-bit parity problems and the **cancer** classification problem of the PROBEN1 set (the standard PROBEN1 benchmarking rules were applied). We have also tried to highlight the benefits that can arise either solely from our method (which is useful in the vicinity of flat minima) or combined with other well-known and effective learning algorithms. To this end, we report performance results for DCBP and the following learning algorithms: back-propagation with momentum (BPM), resilient propagation (RPROP) (Riedmiller & Braun, 1993), ALECO-2 (Perantonis & Karras, 1995), and conjugate gradient methods of Fletcher–Reeves (CG/FR), and Polak–Ribière (CG/PR) with restarts (Johansson, Dowla & Goodman, 1992).

For each of the 3-bit and 4-bit parity problems we performed 100 trials with various initializations of the weights in the range -0.1 to 0.1. The maximum number of epochs per trial was set to 1000 and learning was considered successful when Fahlman's (Fahlman, 1988) '40–20–40' criterion was met. For the 3-bit parity problem we used a 3–3–1 network whereas for the 4-bit parity problem we used a 4–4–1 network. Learning parameters chosen to ensure the best possible performance of each algorithm and training results are shown in Table 1 for the 3-bit problem and in Table 2 for the 4-bit problem. Note that CG/PR and CG/FR do not have any adjustable parameters since an exact line minimization is performed along the direction found at each iteration. From these tables, it is evident that DCBP is able to solve the problems with a high success rate in a relatively small number of epochs, due to its ability to avoid getting trapped in flat minima which are very prominent in parity problems.

Table 2
Experimental results as in Table 1 for the 4-bit parity problem

Algorithms	DCBP	BPM	ALECO-2	RPROP	CG/PR	CG/FR
Parameters	$\delta P = 0.2$ $\xi = 0.5$ $\eta = 0.7$ $\alpha = 0.9$ $T_0 = 3.0$	$\eta = 0.7$ $\alpha = 0.5$	$\delta P = 0.3$ $\xi = 0.85$	$\eta^+ = 1.2$ $\eta^- = 0.5$ $\Delta_{max} = 1.0$ $\Delta_{min} = 10^{-6}$ $\Delta_0 = 0.1$		
# epochs	236	660	329	462	150	327
Successes (%)	90.0	7.0	94.0	20.0	30.0	30.0

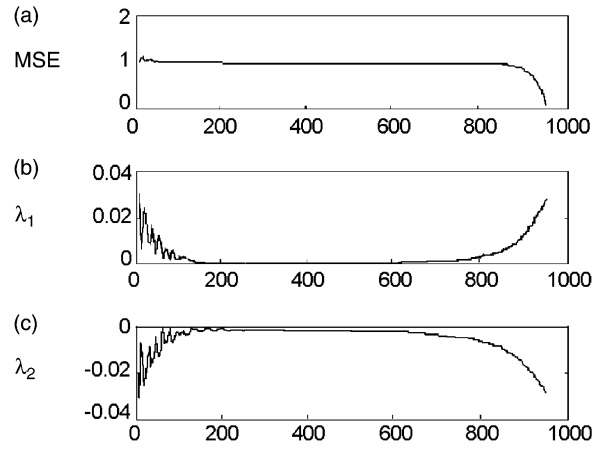

Fig. 1. Plots of mean square error, maximum and minimum Jacobian eigenvalue versus number of epochs for the 3-bit parity problem solved using backpropagation.

In addition, we show the dynamical behavior of each network used when trained both with standard back-propagation (including momentum) and with DCBP. Fig. 1 shows a representative behavior of the dynamics of a 3–3–1 network trained with back-propagation for the 3-bit parity problem. Fig. 1(a) shows the plot of the mean square error (MSE) versus epoch, while Fig. 1(b) and (c) show the plot of the maximum and minimum eigenvalues of the Jacobian matrix, respectively, versus epoch. From Fig. 1(a), the flat minimum corresponding to the initial symmetric state of the network can be characteristically recognized as the part of the MSE curve that is approximately flat. In addition, from Fig. 1(b) and (c) it is clear that as long as the network remains in the vicinity of the flat minimum the two eigenvalues are very small. In particular, the small magnitude of the maximum eigenvalue shown in Fig. 3(b) reveals that the network is unable to move away rapidly from the critical point. Bifurcation of the eigenvalues is clearly seen in Fig. 1(b) and (c) and corresponds to the part of Fig. 1(a) where the network abandons the flat part of the MSE curve.

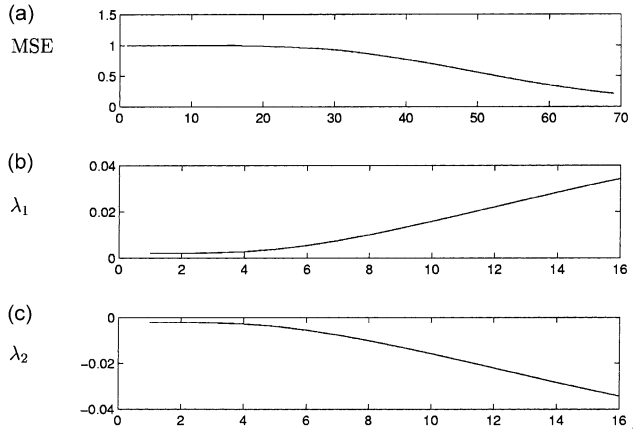

Fig. 2. Plots as in Fig. 1 for the 3-bit parity problem solved using the proposed algorithm.

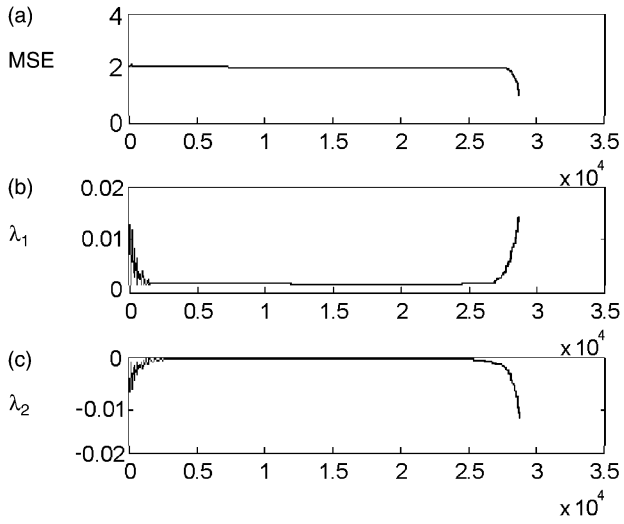

Fig. 3. Plots as in Fig. 1 for the 4-bit parity problem solved using back-propagation.

Fig. 2 shows a representative behavior of the dynamics of the 3–3–1 network trained with the constrained optimization method for the 3-bit parity problem. From the MSE curve, we can see that the learning behavior of the system is altered considerably, since the flat part of the curve is significantly reduced. Fig. 2(b) and (c) clearly show the bifurcation of the eigenvalues achieved with the proposed algorithm within the first few epochs, which with the addition of the alignment with the maximum eigenvector, achieves the fast evolution of the dynamics of the system towards a solution of the problem.

Fig. 3 shows a representative behavior of the dynamics of a 4–4–1 network trained with back-propagation for the 4-bit parity problem. It is interesting to note the extensive initial plateau shown in Fig. 3(a). From Fig. 3(b) and (c), it is evident that the network is able to escape from the minimum only when the eigenvalues of the Jacobian matrix bifurcate. In this particular trial luckily the network did not encounter any further flat minima so that the first redundancy breaking was sufficient for the convergence to the final solution. This was not the case, however, for a different trial shown in Fig. 4 (i.e. with different initial weights) which corresponds to a representative behavior of the dynamics of the 4–4–1 network trained with the constrained optimization method for the 4-bit parity problem. From this figure, we can see that by employing the constrained optimization method, within the first few epochs, we have been able to greatly reduce the number of epochs spent in the initial flat minimum by causing an early bifurcation of the eigenvalues. Following this rapid escape, the network encountered a further flat minimum corresponding to a situation where two hidden nodes formed a particular cluster and, hence, became redundant. Again, the application of the constrained optimization method for a few epochs helped the network to abandon this further minimum and find its way down towards the final solution.

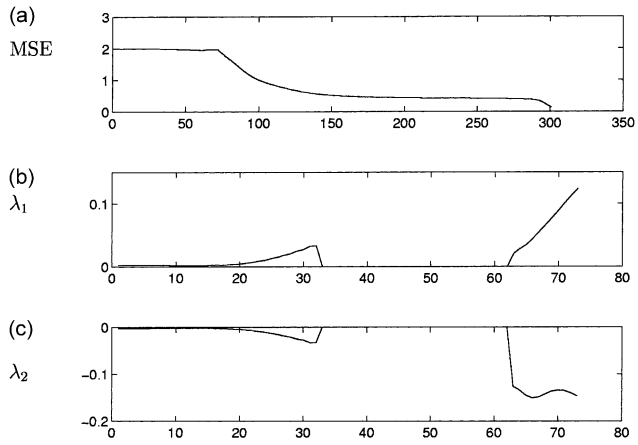

Fig. 4. Plots as in Fig. 1 for the 4-bit parity problem solved using the proposed algorithm.

The **cancer** problem concerns the diagnosis of breast cancer. The task is to classify a tumor as benign or malignant based on cell descriptions gathered by a microscope. The problem has nine real valued inputs, two binary outputs and consists of 699 examples partitioned in 350 training examples, 175 validation examples and 174 test examples. We used the **cancer3** problem dataset of the PROBEN1 database which corresponds to a certain assignment of the samples to each of the three partitions. For this problem, we trained a 9–5–2 network with the training examples, and we performed 10 trials with various initializations of the weights in the range -0.1 to 0.1. The maximum number of epochs per trial was set to 5000 and learning was again considered successful when Fahlman's criterion was met. Learning parameters chosen to ensure the best possible performance of each algorithm and training results are shown in Table 3. From this table, we can make the following interesting observations.

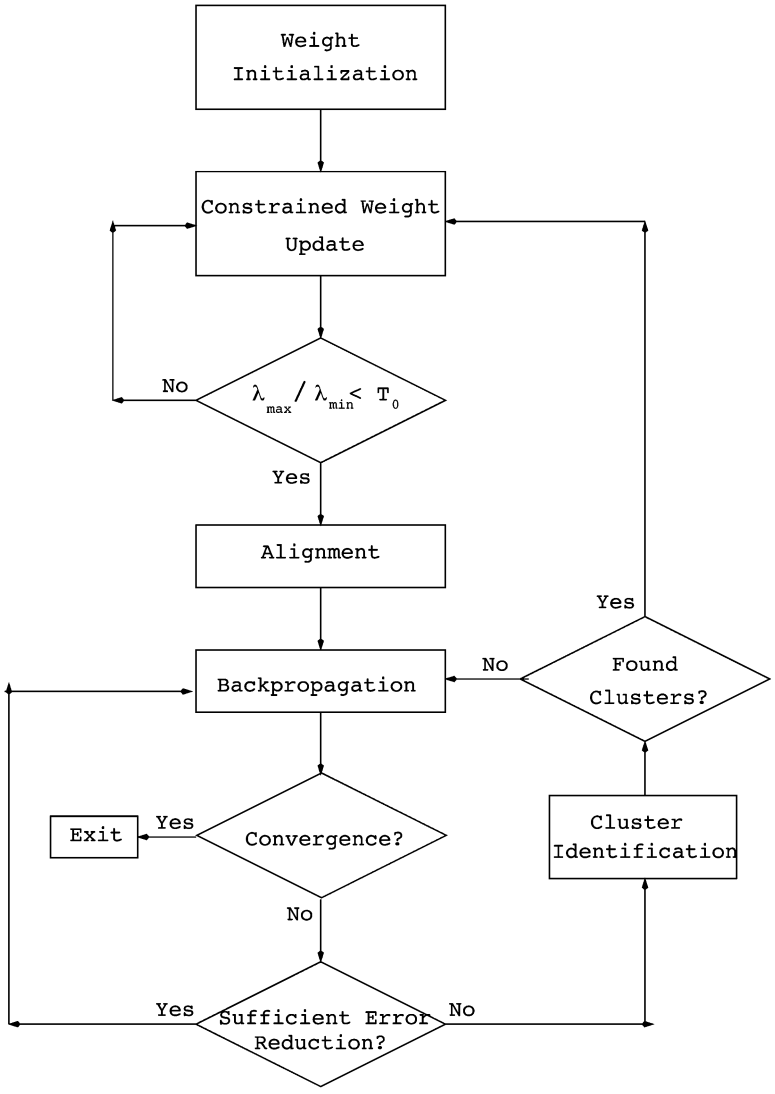

Fig. 4. (*continued*)

Table 3
Experimental results as in Table 1 for the **cancer** problem

Algorithms	DCBP	BPM	ALECO-2	RPROP	CG/PR	CG/FR
Parameters	$\delta P = 0.2$ $\xi = 0.5$ $\eta = 0.15$ $\alpha = 0.15$ $T_0 = 2.0$	$\eta = 0.15$ $\alpha = 0.15$	$\delta P = 0.3$ $\xi = 0.85$	$\eta^+ = 1.2$ $\eta^- = 0.5$ $\Delta_{max} = 1.0$ $\Delta_{min} = 10^{-6}$ $\Delta_0 = 0.1$		
# epochs	3272	Failed	979	346	Failed	Failed
Successes (%)	100.0	0.0	100.0	20.0	0.0	0.0

1. DCBP and ALECO-2 are the only algorithms that exhibit 100% success rate with ALECO-2 requiring a sufficiently smaller mean number of epochs. This is an expected result since both DCBP and ALECO-2 have the potential to escape from flat minima (ALECO-2 achieves this using an adaptive momentum term), but DCBP is slower because when there are no clusters of hidden nodes DCBP switches to BPM which is well known to be slow in valleys and ridges of the cost-function landscape.
2. BPM, CG/PR and CG/FR failed to solve the problem in all the trials, and only RPROP exhibited a small mean number of epochs, but with an unacceptably small success rate. It was suspected that the poor performance of the algorithms was due to the fact that the weights were initialized to small values which is an essential part of DCBP training. Therefore, in order to be fair to these algorithm we performed an additional number of 20 trials. In 10 of these trials the weight initialization was set in the range -0.5 to 0.5, whereas for the other 10 trials the weights were initialized in the range -1.0 to 1.0. The training results of the first 10 trials are shown in Table 4 and the training results of the second 10 trials are shown in Table 5. Learning parameters were the same as those shown in Table 3. From these tables, it is evident that even though all the algorithms were able to solve the problem in some of the trials within a reasonable amount of epochs, the low success rates reveal the potential problem exhibited by most of these algorithms, to get trapped in flat minima.

Regarding computational complexity issues, we have to note the following: in all benchmarks tried, about 15% of the total number of epochs was spent in the vicinity of flat minima with the algorithm operating in the constrained optimization mode, while in the other 85% of the total number of epochs the BPM update rule was used. Moreover, because of the reduction in the number of active variables, the computational overhead for performing a constrained optimization update, relative to performing a standard BPM update, is not particularly heavy. In all benchmarks, the CPU time needed to perform an update in the constrained optimization mode was measured in the range of 2–3 times the CPU time needed to perform an ordinary BPM update. Finally, the clustering algorithm that is employed in order to identify redundant hidden nodes is very fast, with negligible computational cost relative to the gradient evaluation needed to perform a standard BPM update. As a result, we can state that in all our benchmarks, the average computational cost per epoch for our algorithm is in the range of 1.15–1.3 times the corresponding computational cost of BPM. This is an acceptable computational cost compared, for example, to the average cost of the CG method introduced because of the iterative line minimization process (which in the same benchmarks was in the range of 5–8 times the computational cost of BPM).

It is worth noting that as long as a learning algorithm is not trapped in a flat minimum due to hidden node redundancy, its rate of convergence is related to its ability to overcome problems related to the shape of the cost function landscape. For example, DCBP would be able to provide better performance results if at the point where no further clusters of hidden neurons could be detected, one would continue training with a more robust algorithm than BPM. Conversely, other algorithms could benefit by first employing DCBP to eliminate redundancy and then continue further training using their particular characteristics.

In order to verify these assumptions we performed an additional 10 trials with the same initial weights and algorithm parameters used to obtain the results of Table 3. For these trials, we let DCBP run until it reported that it was unable to find any further clusters of hidden neurons. For the **cancer3** problem, this was usually achieved within the first

Table 4
Experimental results for the **cancer** problem. Performance of standard algorithms with initial weights between -0.5 and 0.5

Algorithms	BPM	ALECO-2	RPROP	CG/PR	CG/FR
# epochs	2598	712	404	317	409
Successes (%)	10.0	70.0	10.0	30.0	20.0

Table 5
Experimental results for the **cancer** problem. Performance of standard algorithms with initial weights between -1.0 and 1.0

Algorithms	BPM	ALECO-2	RPROP	CG/PR	CG/FR
# epochs	2692	487	503	300	1488
Successes (%)	80.0	60.0	30.0	20.0	40.0

Table 6
Experimental results for the **cancer** problem. DCBP followed by each of the standard algorithms

Algorithms	DCBP + ALECO-2	DCBP + EPROP	DCBP + CG/PR	DCBP + CG/FR
Parameters	$\delta P = 0.3$ $\xi = 0.5$	$\eta^+ = 1.2$ $\eta^- = 0.5$ $\Delta_{max} = 1.0$ $\Delta_{min} = 10^{-6}$ $\Delta_0 = 0.1$		
# epochs	628	914	452	689
Successes (%)	100.0	10.0	80.0	90.0

300 epochs. From that point on, we continued training using all the other learning algorithms except BPM (which is the usual algorithm following DCBP). The results are shown in Table 6. From this table it is evident that the performance of most of the algorithms was greatly enhanced, with the exception of RPROP which had the tendency to end up in local minima for most of the trials of this particular problem. DCBP followed by ALECO-2 succeeded in all trials with the smallest mean number of epochs, whereas CG/PR and CG/FR exhibited remarkable improvement using this arrangement while they had failed in all trials when they were let to run alone from the same initial weights.

6. Conclusions

In this paper, a dynamical system model for feedforward networks has been introduced. The model is useful for analyzing the dynamics of learning in feedforward networks in the vicinity of flat minima arising from redundancy of nodes in the hidden layer. It was shown that, as a direct consequence of the build up of redundancy, it is possible to describe the dynamics of feedforward networks using appropriate state variables whose total number is reduced compared to the total number of free network parameters (weights and biases). For off-line learning, progress in the objective of abandoning the vicinity of flat minima is related to the magnitude of the largest eigenvalues of each Jacobian matrix associated with a cluster of redundant hidden nodes. Following the onset of bifurcation of the eigenvalues, the network escapes from the flat minimum with components of its weight vector approximately aligned to components of eigenvectors of the Jacobian matrix corresponding to their maximum eigenvalues. This information has been taken into account for proposing a learning algorithm (DCBP) which is able to identify clusters of redundant hidden nodes as they are formed. The algorithm is based on constrained optimization methods and its aim is to drive the state of the network away from the critical points caused by redundancy. This is achieved by maximization of the largest eigenvalues of the Jacobian matrices so that the bifurcation is accelerated and the network is able to escape from flat minima.

There are several research issues pertaining to this novel approach to the dynamics of back-propagation networks. In our opinion, the identification of the Jacobian of the dynamical system which has been accomplished in this paper may provide a useful analytical tool for providing answers, accompanied by proofs, to important theoretical issues. An interesting problem which we are working on is to show analytically that stationary points of the cost function caused by redundancy in a general feedforward network are indeed flat minima (saddle points) and not true local minima. It is possible that this trend of research would advance knowledge on the nature of stationary points in feedforward networks beyond the current state-of-the-art, whereby complete analytical results exist only for specific small scale networks and tasks such as the XOR problem (Hamey, 1998; Lisboa & Perantonis, 1991; Sprinkhuizen-Kuyper & Boers, 1996a,b). From the algorithmic point of view, there is scope for further improvement. Future plans of our research involve techniques for the automatic adaptation of the parameters δP, $\boldsymbol{\xi}$ and T_0 at each iteration of the algorithm which will result in a powerful, fully automated method requiring minimal input from the end user of the network. In addition, the quantities $\boldsymbol{\Phi}_c$ that are maximized in DCBP constitute only lower bounds to the maximum eigenvalues of the Jacobian matrices, which may be rather loose bounds in larger scale problems. It may be possible to further improve learning capabilities by providing more accurate estimates for the evolution of the eigenvalues. We have already implemented such a technique in small networks by employing theoretical tools from matrix perturbation theory (Ampazis, Perantonis & Taylor, 1999b) with very promising results. Finally, the implications of our dynamical analysis to on-line learning methods such as those based on information geometry is currently under investigation and we hope that we will be able to report soon on the corresponding results.

References

Amari, S. (1985). Differential-geometrical method in statistics. *Springer Lecture Note in Statistics*, 28.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*, 251–276.

Amari, S., & Nagaoka, H. (2000). *Methods of information geometry*, AMS.

Amari, S., Park, H., & Fukumizu, K. (1999). Adaptive method of realizing

natural gradient learning for multilayer perceptrons. *Neural Computation*, *12*, 1399–1409.

Ampazis, N., Perantonis, S. J., & Taylor, J. G. (1999a). Dynamics of multilayer networks in the vicinity of temporary minima. *Neural Networks*, *12* (1), 43–58.

Ampazis, N., Perantonis, S. J., & Taylor, J. G. (1999). Acceleration of learning in feed-forward networks using dynamical systems analysis and matrix perturbation theory. In *Proceedings of International Joint Conference on Neural Networks*. Washington, DC.

Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms*, New York: Plenum Press.

Chiu, S. (1994). Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, *2*(3), 269–278.

Fahlman, S. E. (1988). Faster learning variations on back-propagation: an empirical study. In D. Touretsky, G. Hinton & T. Sejnowski, *Proceedings of the Connectionist Models Summer School* (pp. 38–51). San Mateo: Morgan Kaufmann.

Fukumizu, K., & Amari, S. (2000). Local minima and plateaus in hierarchical structures of multi-layer perceptrons. *Neural Networks*, *13*, 317–327.

Hamey, L. (1998). XOR has no local minima: a case study in neural network error surface analysis. *Neural Networks*, *11*, 669–681.

Johansson, E. M., Dowla, F. U., & Goodman, D. M. (1992). Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, *2* (4), 291–301.

Lisboa, P. J. G., & Perantonis, S. J. (1991). Complete solution of the local minima in the XOR problem. *Network*, *2*, 119–124.

Perantonis, S. J., & Karras, D. A. (1995). An efficient constrained learning algorithm with momentum acceleration. *Neural Networks*, *8*, 237–249.

Prechelt, L. (1994). *PROBEN1—a set of neural network benchmark problems and benchmarking rules*. Technical Report 21/94, Universität Karlsruhe, Germany.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *Proceedings of the International Conference on Neural Networks* (vol. 1, pp. 586–591). San Francisco.

Sprinkhuizen-Kuyper, I. G., & Boers, E. J. W. (1996a). The error surface of the simplest XOR network has only global minima. *Neural Computation*, *8*, 1301–1320.

Sprinkhuizen-Kuyper, I. G., & Boers, E. J. W. (1996). *The error surface of the 2–2–1 XOR network: stationary points with infinite weights*. Technical Report 96-10, Department of Computer Science, Leiden University.

Yager, R., & Filev, D. (1994). Generation of fuzzy rules by mountain clustering. *Journal of Intelligent and Fuzzy Systems*, *2* (3), 209–219.

Yang, H. H., & Amari, S. (1998). Complexity issues in natural gradient descent method for training multilayer perceptrons. *Neural Computation*, *10*, 2137–2157.