A Constrained Learning Framework for Feedforward Neural Networks

Stavros J. Perantonis [a,*], Nikolaos Ampazis [a] and Vassilis Virvilis [a]

[a] *Institute of Informatics and Telecommunications, National Center for Scientific Research "Demokritos", 153 10 Agia Paraskevi, Greece*
E-mail: sper@iit.demokritos.gr

Conventional supervised learning in neural networks is carried out by performing unconstrained minimization of a suitably defined cost function. This approach has certain drawbacks, which can be overcome by incorporating additional knowledge in the training formalism. In this paper, two types of such additional knowledge are examined: Network specific knowledge (associated with the neural network irrespectively of the problem whose solution is sought) or problem specific knowledge (which helps to solve a specific learning task). A constrained optimization framework is introduced for incorporating these types of knowledge into the learning formalism. We present three examples of improvement in the learning behaviour of neural networks using additional knowledge in the context of our constrained optimization framework. The two network specific examples are designed to improve convergence and learning speed in the broad class of feedforward networks, while the third problem specific example is related to the efficient factorization of 2-D polynomials using suitably constructed sigma-pi networks.

Keywords: Neural networks, supervised learning, constrained optimization

1. Introduction

Methods from the field of Optimization have played an important role in developing training algorithms for connectionist systems. Indeed, the realization that simple gradient descent (back propagation - BP) can be applied with success to the training of multilayered feedforward networks (MFNs) [1] was responsible to a great extent for the resurgence of interest in this kind of networks during the mid 1980s. Most of the methods used for supervised learning originate from

* corresponding author

unconstrained optimization techniques. Obviously, this is related to the "black box" nature of connectionist systems: Apart from the minimization of a cost function, no other information or knowledge is usually taken into account.

Nevertheless, recent research has shown that it is often beneficial to incorporate additional knowledge in the neural network architecture or learning rule [2]-[6]. Often, the additional knowledge can be encoded in the form of mathematical relations that have to be satisfied in addition to the demand for minimization of the cost function. A number of learning algorithms have been proposed that incorporate "target conditions", i.e. additional conditions to be satisfied upon termination of learning. Notable examples include matching of hidden unit outputs with prespecified targets [7][8], satisfaction of box constraints for the synaptic weights that prevent hidden node saturation [9] or conditions that enable weight decay for the improvement of generalization properties [10][11]. Naturally, methods from the field of constrained optimization are essential for solving these modified neural network learning tasks. In the literature, the most common method for handling extra constraints is the penalty function formulation [10]. Recently, the application of interior point methods has also been investigated [9].

Apart from conditions that must be satisfied as best as possible upon termination of learning (target conditions), experience in training of feedforward networks shows that it is also useful to incorporate additional "incremental conditions", i.e. conditions involving quantities that must be optimized incrementally at each epoch of the learning process. In this paper, we introduce a general constrained optimization framework for incorporating additional knowledge in the neural network learning rule. We thus formulate a general problem, whereby minimization of the cost function representing the distance of the networks outputs from preset target values is sought, subject to other relations that represent the additional knowledge. We show how to formulate the general problem of incorporating additional knowledge as a multiobjective optimization task, whose solution leads to a generic learning algorithm accounting for both target and incremental conditions.

To elucidate the usefulness of the approach, we present three examples of learning algorithms that incorporate additional knowledge about learning and show that they can be derived using the general framework introduced earlier.

In the first two examples, additional information about the specific type of the neural network, the nature and characteristics of its cost function landscape is used to facilitate learning in broad classes of problems. In the first example,

a successful algorithm originally proposed in [6] is derived using our framework for constrained learning. Here the additional information is incorporated in one extra condition that seeks to facilitate navigation in long narrow valleys of the cost function landscape, thus accelerating learning. In the second example, the objective is to accelerate learning and improve convergence through successful negotiation of the cost function landscape in the vicinity of temporary minima. In previous related work this problem was studied for networks of two hidden nodes using a dynamical systems approach [12]. In this paper, the dynamical systems approach is extended to networks with a hidden layer of an arbitrary number of nodes. This extension provides us with information about the nature of temporary minima, whose successful negotiation can be achieved using the general constrained learning framework with multiple additional objectives.

In the final example, the objective is to solve a specific problem (polynomial factorization) using an appropriate neural network formulation. In previous work, we have used neural networks to solve this problem [13] taking into account extra conditions among the polynomial coefficients. In this paper we show that a neural network learning algorithm within the general constrained learning framework studied in this paper can be used to solve the polynomial factorization problem effectively.

For all examples, we present simulations to compare the performance of the proposed constrained optimization learning method with other state of the art algorithms for multilayered feedforward networks.

2. A generic constrained learning algorithm

Conventional unconstrained supervised learning in neural networks involves minimization, with respect to the synaptic weights and biases, of a cost function of the form

$$E[d_i^{(p)} - y_i^{(p)}(\boldsymbol{w})] \tag{1}$$

Here $\boldsymbol{w}$ is a column vector containing all synaptic weights and biases of the network, p is an index running over the $\mathcal{P}$ patterns of the training set, $y_i^{(p)}$ is the network output corresponding to output node i and $d_i^{(p)}$ is the corresponding target.

In this paper we suppose that there are additional relations to be satisfied, that represent the extra knowledge and involve the network's synaptic weights.

Before introducing the form of the extra relations, we note that in order to minimize the cost function of equation (1) we adopt an epoch-by-epoch optimization framework with the following objectives:

- At each epoch of the learning process, the vector $\boldsymbol{w}$ is to be incremented by $d\boldsymbol{w}$, so that the search for an optimum new point in the space of $\boldsymbol{w}$ is restricted to a hypersphere of known radius δP centered at the point defined by the current $\boldsymbol{w}$:

$$d\boldsymbol{w}^T d\boldsymbol{w} = (\delta P)^2 \tag{2}$$

- At each epoch, the cost function must be decremented by a positive quantity δQ, so that at the end of learning E is rendered as small as possible. To first order, we can substitute the change in E by its first differential and demand that:

$$dE = \delta Q \tag{3}$$

We now wish to introduce additional objectives in order to incorporate the extra knowledge into the learning formalism. We consider the following two cases of importance that involve additional mathematical relations representing knowledge about learning in neural networks:

Case 1: There are additional constraints $\boldsymbol{\Phi} = \mathbf{0}$ which must be satisfied as best as possible upon termination of the learning process. Here $\boldsymbol{\Phi}$ is a column vector whose components are known functions of the synaptic weights. We address this problem by introducing a function κ and demanding at each epoch of the learning process the maximization of κ subject to the condition that $d\boldsymbol{\Phi} = -\kappa\boldsymbol{\Phi}$. In this way, we ensure that $\boldsymbol{\Phi}$ tends to $\mathbf{0}$ at a temporarily exponential rate.

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 $\boldsymbol{w}$:

$$\underset{dw \in R}{\text{Maximize}} \qquad \kappa \tag{4}$$

$$\text{s. t.} \qquad d\boldsymbol{w}^T d\boldsymbol{w} = (\delta P)^2 \tag{5}$$

$$dE = \delta Q \tag{6}$$

$$d\Phi_m/\Phi_m = -\kappa, \quad m = 1, \ldots, S \tag{7}$$

where S is the number of components of $\mathbf{\Phi}$.

Case 2: This case involves additional conditions whereby there is no specific final target for the vector $\mathbf{\Phi}$, but rather it is desired that all components of $\mathbf{\Phi}$ are rendered as large as possible at each individual epoch of the learning process. This is a multiobjective maximization problem, which we address by defining $\kappa = d\Phi_m$ and demanding that κ assume the maximum possible value at each epoch. Thus the constrained optimization problem is as before with equation (7) substituted by

$$\kappa = d\Phi_m, \quad m = 1, \ldots, S \tag{8}$$

The solution to the above constrained optimization problems can be obtained by a method similar to the constrained gradient ascent technique introduced by Bryson and Denham [14] and leads to a generic update rule for $\boldsymbol{w}$. We thus introduce suitable Lagrange multipliers L_1 and L_2 to take account of equations (6) and (5) respectively and a vector of multipliers $\mathbf{\Lambda}$ to take account of equation (7) or equation (8). The Lagrangian thus reads

$$\kappa' = \kappa + L_1(\boldsymbol{G}^T d\boldsymbol{w} - \delta Q) + L_2[d\boldsymbol{w}^T d\boldsymbol{w} - (\delta P)^2] + \mathbf{\Lambda}^T(\boldsymbol{F} d\boldsymbol{w} + \kappa \mathbf{1}) \tag{9}$$

where the following quantities have been introduced:

- A vector $\boldsymbol{G}$ with elements $G_i = \partial E/\partial w_i$
- A matrix $\boldsymbol{F}$ whose elements are defined by $F_{mi} = (1/\Phi_m)(\partial \Phi_m/\partial w_i)$ (for case 1) or $F_{mi} = -\partial \Phi_m/\partial w_i$ (for case 2).

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

$$d\kappa' = d\kappa(1 + \mathbf{\Lambda}^T \mathbf{1}) + (L_1 \boldsymbol{G}^T + \mathbf{\Lambda}^T \boldsymbol{F} + 2L_2 d\boldsymbol{w}^T) d^2\boldsymbol{w} = 0 \tag{10}$$

$$d^2\kappa' = 2L_2 d^2\boldsymbol{w}^T d^2\boldsymbol{w} < 0 \tag{11}$$

Hence, the factors multiplying $d^2\boldsymbol{w}$ and $d\kappa$ in equation (9) should vanish, and therefore we obtain:

$$d\boldsymbol{w} = -\frac{L_1}{2L_2}\boldsymbol{G} - \frac{1}{2L_2}\boldsymbol{F}^T \mathbf{\Lambda} \tag{12}$$

$$\mathbf{\Lambda}^T \mathbf{1} = -1 \tag{13}$$

Equation (12) constitutes the weight update rule for the neural network, provided that the Lagrange multipliers appearing in it have been evaluated in terms of

known quantities. The result of this evaluation is summarized forthwith, whereas the full evaluation is carried out in the Appendix. To complete the evaluation, it is necessary to introduce the following quantities:

$$I_{GG} = \boldsymbol{G}^T \boldsymbol{G} \tag{14}$$

$$\boldsymbol{I_{GF}} = \boldsymbol{F}\boldsymbol{G} \tag{15}$$

$$\boldsymbol{I_{FF}} = \boldsymbol{F}\boldsymbol{F}^T \tag{16}$$

$$\boldsymbol{R} = \frac{1}{I_{GG}}(I_{GG}\boldsymbol{I_{FF}} - \boldsymbol{I_{GF}} \times \boldsymbol{I_{GF}}) \tag{17}$$

$$Z = 1 + \frac{(\boldsymbol{R}^{-1})_a(\boldsymbol{I_{GF}}^T \boldsymbol{R}^{-1} \boldsymbol{I_{GF}}) - (\mathbf{1}^T \boldsymbol{R}^{-1} \boldsymbol{I_{GF}})^2}{(\boldsymbol{R}^{-1})_a I_{GG}} \tag{18}$$

where $(.)_a$ denotes the sum of all elements of a matrix and $\mathbf{1}$ is a column vector whose elements are all equal to 1. In terms of these known quantities, the Lagrange multipliers are evaluated using the relations:

$$L_2 = -\frac{1}{2}\left[\frac{I_{GG}}{(\boldsymbol{R}^{-1})_a[I_{GG}(\delta P)^2 - Z(\delta Q)^2]}\right]^{1/2} \tag{19}$$

$$\boldsymbol{\Lambda} = -\frac{\boldsymbol{R}^{-1}\mathbf{1}}{(\boldsymbol{R}^{-1})_a} - \frac{2L_2\delta Q}{I_{GG}}\left[\frac{\mathbf{1}^T \boldsymbol{R}^{-1}\boldsymbol{I_{GF}}}{(\boldsymbol{R}^{-1})_a}\boldsymbol{R}^{-1}\mathbf{1} - \boldsymbol{R}^{-1}\boldsymbol{I_{GF}}\right] \tag{20}$$

$$L_1 = -\frac{2L_2\delta Q + \boldsymbol{\Lambda}^T \boldsymbol{I_{GF}}}{I_{GG}} \tag{21}$$

In the Appendix, it is shown that δQ must be chosen adaptively according to the relation $\delta Q = -\xi \delta P \sqrt{I_{GG}}$. Here ξ is a real parameter with $0 < \xi < 1$. Consequently, the proposed generic weight update algorithm has two free parameters, namely δP and ξ.

3. Learning in MFNs: Negotiating long valleys

Consider a feedforward network with two layers of weights 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 set of $\mathcal{P}$ training patterns indexed by p, the off-line BP algorithm is obtained by performing gradient descent with respect to the mean square error (MSE) cost function

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{K}(d_i^{(p)} - y_i^{(p)})^2 \tag{22}$$

where $y_i^{(p)}$ denote the output activations and $d_i^{(p)}$ are the desired responses.

Learning in feedforward networks is usually hindered by specific characteristics of the landscape defined by the MSE cost function. The two most common problems arise because

- of the occurrence of long, deep valleys or troughs that force gradient descent to follow zig-zag paths.
- of the possible existence of temporary minima in the cost function landscape.

In cost function landscapes with long deep valleys, back propagation (gradient descent) is highly inefficient because it settles into zig-zag paths and is hopelessly slow [15]. Supplementing gradient descent with a momentum term proportional to the weight update vector of the previous epoch represents a compromise between the need to decrease the cost function and the need to proceed along relatively smooth paths in the weight space. By introducing the momentum term which is proportional to the previous epoch weight update, paths whereby current and previous weight update vectors are partially aligned are favored, smoother trajectories are followed and learning is accelerated. An illustration of this improved behaviour is given in [16] (p. 123). However, the selection of appropriate values for the learning rate and the momentum term coefficient is difficult and a method of adaptively determining these coefficients based on landscape characteristics is highly desirable. To this end, an iterative algorithm was introduced in [6] whose purpose is to maximize, at each epoch, the alignment between the current and previous weight update vectors without compromising the need for decrease of the cost function. This helps achieve the maximum possible alignment of successive weight update vectors, thus further suppressing zig-zagging and accelerating learning. To this end, we require satisfaction of an additional condition, amounting to maximization of the quantity $\Phi = (\boldsymbol{w} - \boldsymbol{w}_t)^T(\boldsymbol{w}_t - \boldsymbol{w}_{t-1})$ with respect to the synaptic weight vector $\boldsymbol{w}$ at each epoch of the algorithm. Here $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t-1}$ are the values of the weight vectors at the present and immediately

preceding epoch respectively and are treated as known constant vectors. Since within our constrained learning framework $\boldsymbol{w} - \boldsymbol{w}_t$ and $\boldsymbol{w}_t - \boldsymbol{w}_{t-1}$ have constant moduli equal to δP (by equation 5), maximization of Φ amounts to minimization of the angle between successive weight update vectors. Once the quantity to be maximized at each epoch has been specified, the learning rule can be derived readily from the generic constrained learning algorithm presented in the previous section. We have only one additional condition to satisfy (maximization of $d\Phi$ at each epoch), so that Case 2 of the previous section is applicable. It is readily seen that in this case $\boldsymbol{\Lambda}$ has only one component equal to -1 (by equation (13)) and the weight update rule is quite simple:

$$d\boldsymbol{w} = -\frac{L_1}{2L_2}\boldsymbol{G} - \frac{1}{2L_2}\boldsymbol{u} \tag{23}$$

where

$$\boldsymbol{u} = \boldsymbol{w}_t - \boldsymbol{w}_{t-1}, \quad L_1 = (I_{uG} - 2L_2\delta Q)/I_{GG}, \quad L_2 = \frac{1}{2}\left[\frac{I_{GG}I_{uu} - I_{uG}^2}{I_{GG}(\delta P)^2 - (\delta Q)^2}\right]^{1/2} \tag{24}$$

with

$$I_{uG} = \boldsymbol{u}^T\boldsymbol{G}, \quad I_{uu} = \boldsymbol{u}^T\boldsymbol{u} \tag{25}$$

This is identical to the learning rule derived in [6]. Hence, weight updates are formed as linear combinations of the cost function derivatives $\boldsymbol{G}$ with respect to the weights and of the weight updates $\boldsymbol{u}$ at the immediately preceding epoch. This is similar to back propagation with a momentum term, with the essential difference that the coefficients of $\boldsymbol{G}$ and $\boldsymbol{u}$ are suitably adapted at each epoch of the learning process.

The resulting learning algorithm is herewith compared with BP and a host of other well known algorithms that have been used by other authors to train feedforward networks. Simulations quoted here and in the next section were conducted using Billnet, a locally developed neural network simulator (available at `http://www.iit.demokritos.gr/~vasvir`). We report performance results for two well known benchmarks. In particular, results for the 11-11-1 multiplexer problem with 2048 patterns [17] are shown in Table 1, whereas results for the

	Proposed (sect. 3)	BPM off	BPM on	RPROP	CG (PR)	CG (FR)	QP	DBD
	δP:0.8 ξ:0.5	NC	η:0.6 α:0.7	η^+:1.2 η^-:0.5 Δ_M:1.0 Δ_m:10^{-6} Δ_0:0.1			NC	NC
Epochs (Mean)	145	F	287	284	148	F	F	F
Epochs (StD)	38	F	95	79	35	F	F	F
Successes (%)	100	0	94	90	70	0	0	0

Table 1

Results in terms of number of epochs (mean value and standard deviation) and success rates for the 11-11-1 multiplexer task using the constrained learning algorithm of section 3 and other well known algorithms. F and NC denote failure of convergence in all trials.

64-8-64 encoder problem with 64 patterns [18] are given in Table 2. We assess the performance of a much wider range of algorithms than those tested in [6], namely the on-line and off-line versions of BP with momentum (BPM), resilient propagation (RPROP) [19], conjugate gradient methods of Fletcher-Reeves (CG/FR) and Polak-Ribiére (CG/PR) with restarts [20], the quickprop algorithm of Fahlman (QP) [18] and the Delta Bar Delta algorithm of Jacobs (DBD) [17]. For each benchmark problem we performed 50 learning trials starting from different randomly chosen weights in the range -0.5 to 0.5. The maximum number of epochs per trial was set to 1000 and learning was considered successful when Fahlman's "40-20-40" criterion was met [18]. Learning parameters were chosen to ensure the best possible performance for each algorithm. Symbols used for the parameters in Tables 1 and 2 are the same used by the corresponding authors. The average and standard deviation of the number of epochs in the 50 trials is quoted, along with the success rate, i.e. the percentage of trials for which learning was successful according to Fahlman's criterion. Note that the proposed algorithm is the only method fully successful in all trials for both benchmarks. Moreover, it exhibits the lowest average number of epochs in the successful trials, comparable only to the average number of epochs achieved by CG/PR. In all methods, most computational burden lies in the forward pass (for all patterns) and in the backward pass through the MFN (both of $O(\mathcal{P}\mathcal{N}_{\sqsupseteq})$ where N_w is the total number of weights and biases). For the multiplexer and encoder benchmarks considered here, CG utilized an average of 35 forward passes through the network per epoch

	Proposed (sect. 3)	BPM off	BPM on	RPROP	CG (PR)	CG (FR)	QP	DBD
	δP: 1.5 ξ: 0.5	NC	η: 0.5 α: 0.7	NC			ϵ: 4.0 μ: 1.75 ω: -10^{-4} α: 0.0	NC
Epochs (Mean)	161	F	285	F	170	F	410	F
Epochs (StD)	15	F	40	F	30	F	150	F
Successes (%)	100	0	100	0	100	0	98	0

Table 2

Results in terms of number of epochs (mean value and standard deviation) and success rates for the 64-6-64 encoder benchmark learning task using the constrained learning algorithm of section 3 and other well known algorithms.

during the line minimization phase. This is to be added to the cost of one forward and one backward pass needed in the gradient evaluation phase. On the other hand, all other algorithms examined here, including the proposed algorithm, just involve one forward and one backward pass per epoch.

4. Dynamics of learning near temporary minima

As mentioned in the previous section, the problem of slow learning in feedforward networks is also associated with the problem of temporary minima in the cost function landscape. In recent work [12], we have studied the problem of temporary minima in feedforward networks with just two hidden nodes using a method that originates from the theory of dynamical systems. One of the major results obtained are the analytical predictions for the characteristic dynamical transitions from flat plateaus (or temporary minima) of finite error to the desired levels of lower or even zero error. Our results have taken the form of closed dynamical laws for a finite set of characteristic observables describing the dynamical process of learning and the various transitions involved for MFNs. The analysis carried out in [12] can be generalized to networks with an arbitrary number of nodes in the hidden layer, as explained in the rest of this section. Moreover, results of the analysis can be incorporated into the general constrained learning

framework of section 2 in order to improve the learning behaviour in feedforward networks.

Consider an MFN with one layer of hidden nodes. Given a small learning rate, the difference weight update equations of the off-line BP algorithm can be approximated by differential equations in time. The weight update rule for the hidden-to-output connections gives:

$$\dot{w}_{ij} = \sum_p (d_i^{(p)} - y_i^{(p)}) y_i^{(p)} (1 - y_i^{(p)}) y_j^{(p)} \tag{26}$$

where w_{ij} are the weight connections between each hidden node j and output node i, and $y_j^{(p)}$ are the outputs of each hidden unit (with $j = 0$ corresponding to the bias signal).

For each hidden node j, the off-line BP rule for the input-to-hidden connections reads as follows:

$$\dot{\boldsymbol{w}}_j = \sum_p \sum_i (d_i^{(p)} - y_i^{(p)}) y_i^{(p)} (1 - y_i^{(p)}) w_{ij} y_j^{(p)} (1 - y_j^{(p)}) \boldsymbol{x}^{(p)} \tag{27}$$

where

$$\boldsymbol{w}_j = (w_{j0} \ldots w_{jN})^T \tag{28}$$

with w_{jk} representing the weight connection between hidden node j and input node k, and $\boldsymbol{x}^{(p)}$ denoting the input pattern vector.

It is well known that temporary minima result from the development of internal symmetries and from the subsequent building of redundancy in the hidden layer. In this case, one or more of the hidden nodes perform approximately the same function and therefore they form clusters of redundant nodes. Due to the formation of these clusters, the network is trapped in a temporary minimum and it usually takes a very long time before the redundancy is broken and it finds its way down the cost function landscape.

For a two-layer network, consider a number M_c of hidden units that form a particular cluster C in the vicinity of a temporary minimum. Because of the building of redundancy, all hidden units in C perform approximately the same function and this is reflected in a near equality of synaptic weights leading to or emanating from them (equality is strict exactly at the temporary minimum and approximate in its immediate vicinity). Based on this observation, we can introduce suitable state variables formed by appropriate linear combinations of the synaptic weights, and derive a dynamical system model which describes the

dynamics of the feedforward network in the vicinity of these temporary minima. We expect that appropriate state variables that map the temporary minimum to the origin of the phase plane are formed using the weights connected to hidden nodes j belonging to C. In particular, we can consider the following state variables:

- the differences of each weight vector $\boldsymbol{w}_j$ from the average of all $\boldsymbol{w}_j$ $(j \in C)$
- the differences of each weight w_{ij} from the average of all weights w_{ij} $(j \in C)$.

Hence we define

$$\boldsymbol{\omega}^c = \frac{\sum_{j \in C} \boldsymbol{w}_j}{M_c} \tag{29}$$

$$\boldsymbol{\epsilon}_j = \boldsymbol{w}_j - \boldsymbol{\omega}^c, \quad j \in C \tag{30}$$

It follows that

$$\dot{\boldsymbol{\epsilon}}_j = \dot{\boldsymbol{w}}_j - \dot{\boldsymbol{\omega}}^c = \dot{\boldsymbol{w}}_j - \frac{\sum_{j \in C} \dot{\boldsymbol{w}}_j}{M_c} \tag{31}$$

Similarly for each output node i, we define

$$\nu_i^c = \frac{\sum_{j \in C} w_{ij}}{M_c} \tag{32}$$

$$\mu_{ij} = w_{ij} - \nu_i^c, \quad j \in C \tag{33}$$

Hence

$$\dot{\mu_{ij}} = \dot{w_{ij}} - \dot{\nu_i^c} = \dot{w_{ij}} - \frac{\sum_{j \in C} \dot{w_{ij}}}{M_c} \tag{34}$$

Using equations (26) and (27) and keeping only first order terms in the state variables μ_{ij} and ϵ_j, we can obtain from equations (34) and (31) the following linear differential equations that describe the dynamics of the system:

$$\dot{\mu_{ij}} = \sum_p (d_i^{(p)} - y_i^{0(p)}) y_i^{0(p)} (1 - y_i^{0(p)}) f'(\boldsymbol{\omega}^c \cdot \boldsymbol{x}^{(p)}) (\boldsymbol{\epsilon}_j \cdot \boldsymbol{x}^{(p)}) \tag{35}$$

$$\begin{aligned} \dot{\boldsymbol{\epsilon}_j} = & \sum_{pi} (d_i^{(p)} - y_i^{0(p)}) y_i^{0(p)} (1 - y_i^{0(p)}) f'(\boldsymbol{\omega}^c \cdot \boldsymbol{x}^{(p)}) \\ & \cdot \boldsymbol{x}^{(p)} \{\mu_{ij} + \nu_i [1 - 2 f(\boldsymbol{\omega}^c \cdot \boldsymbol{x}^{(p)})] (\boldsymbol{\epsilon}_j \cdot \boldsymbol{x}^{(p)})\} \end{aligned} \tag{36}$$

where $f' = f(1 - f)$ and

$$y_i^{0(p)} = f[M_c \nu_i^c f(\boldsymbol{\omega}^c \cdot \boldsymbol{x}^{(p)}) + \sum_{J \notin C} w_{iJ} y_J^{(p)}] \tag{37}$$

From equations (36) and (35) we observe that time derivatives of state variables corresponding to a certain hidden node j depend only on state variables corresponding to the same hidden node. Therefore, introducing the vector $\boldsymbol{u}_j = (\boldsymbol{\epsilon}_j^T, \mu_{ij})^T$, $i = 1, \ldots, K$ we obtain M_c equations of the form:

$$\dot{\boldsymbol{u}}_j = \boldsymbol{J}_c \boldsymbol{u}_j \tag{38}$$

The Jacobian matrix $\boldsymbol{J}_c$ is given by

$$\boldsymbol{J}_c = \sum_p f'(\boldsymbol{\omega}^c \cdot \boldsymbol{x}^{(p)}) \cdot \begin{pmatrix} \sum_i A_i^{(p)} [1 - 2f(\boldsymbol{\omega}^c \cdot \boldsymbol{x}^{(p)})] \boldsymbol{x}^{(p)} \boldsymbol{x}^{(p)T} \nu^c & \boldsymbol{A}^{(p)} \boldsymbol{x}^{(p)} \\ (\boldsymbol{A}^{(p)} \boldsymbol{x}^{(p)})^T & \boldsymbol{0} \end{pmatrix} \tag{39}$$

where

$$\boldsymbol{A}^{(p)} = \left((d_i^{(p)} - y_i^{0(p)}) y_i^{0(p)} (1 - y_i^{0(p)}),\ i = 1, \ldots, K \right) \tag{40}$$

Note that the dimension of all vectors $\boldsymbol{u}_j$ is $D = N + K + 1$ and therefore all Jacobian matrices are of dimension $D \mathrm{x} D$ (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. Due to the exponential nature of the solutions, the followed trajectory obviously depends on the magnitude of the largest eigenvalue for each Jacobian matrix. Initially, in the vicinity of the temporary minimum all eigenvalues are small in magnitude and the network spends a relatively long time in the vicinity of the critical point. As time passes, the magnitude of the largest eigenvalues grows, so that eventually a bifurcation of the eigenvalues occurs and the system follows a trajectory which allows it to move far away from the critical point. In citeTaylor specific examples of this behaviour in networks with just two hidden nodes are given. Asymptotically, the trajectories followed by the dynamical systems described by equation (38) are then 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:

$$d\boldsymbol{u}_j \approx \mathrm{sign}(\boldsymbol{u}_j^T \boldsymbol{\xi}^c)\ \boldsymbol{\xi}^c \tag{41}$$

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

Following this analysis, it is evident that if the maximum eigenvalues $\lambda_c, c = 1, \ldots S$ of the Jacobian matrices $\boldsymbol{J}_c$ corresponding to each of the S clusters of hidden nodes are relatively large, then the network is able to escape from the

temporary minimum. Hence, instead of waiting for the growth of the eigenvalues, the objective of our approach is to raise these eigenvalues more rapidly in order to facilitate learning. It is therefore beneficiary to incorporate in the learning rule additional knowledge related to the desire for rapid growth of these eigenvalues. 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 $\Phi_c \leq \lambda_c$ for these eigenvalues, which are obtained as follows: It is well known from linear algebra that since $\boldsymbol{J}_c$ is a real and symmetric matrix, then

$$\boldsymbol{z}^{\boldsymbol{T}} \boldsymbol{J}_c \boldsymbol{z} \leq \lambda_c \;\; \boldsymbol{z}^{\boldsymbol{T}} \boldsymbol{z} \quad \forall \boldsymbol{z} \in R^Q \tag{42}$$

We have used the simplest choice for $\boldsymbol{z}$ namely $\boldsymbol{z} = \mathbf{1} = (1 \;\; 1 \ldots 1)^T$ which means that the product in the left hand side of equation (42) is simply the sum of the elements of the matrix.

Therefore we can apply the generic weight update rule (Case 2 of section 2) using

$$\Phi_c = \mathbf{1}^T \boldsymbol{J}_c \mathbf{1} \tag{43}$$

During learning, different clusters of hidden nodes may be formed and the number and structure of the Φ_c must be changed accordingly. Therefore, for the implementation of the algorithm, it is essential to have a method of detecting cluster formation during learning. This detection is performed periodically (at regular intervals of typically 50 epochs) using a standard subtractive clustering algorithm [21] which identifies the number of formed clusters and, therefore, the number of Φ_c. Hidden nodes are then assigned to a cluster using the fuzzy C-means algorithm [22], whereupon the analytical expressions for all Jacobian matrices and the corresponding Φ_c are constructed.

The proposed method is herewith compared with BP and a host of other well known algorithms that have been used by other authors to train feedforward networks. We report performance results for two benchmarks. The first benchmark is the well known 4 bit parity problem, which is characterized by the presence of temporary minima in its cost function. A network with a 4-4-1 architecture is employed to solve the problem. Fifty learning trials were performed starting from different randomly chosen weights in the range -0.5 to 0.5 and the maximum allowed number of epochs per trial was set to 1000. As before, learning was considered successful when Fahlman's "40-20-40" criterion was met. In

	Proposed (sect. 4)	BPM OFF	BPM ON	RPROP	CG (PR)	CG (FR)	QP	DBD
	$\delta P: 0.2$	$\eta: 0.7$	$\eta: 0.7$	$\eta^+: 1.2$			$\epsilon: 1.0$	$\epsilon: 5.0$
	$\xi: 0.5$	$\alpha: 0.5$	$\alpha: 0.9$	$\eta^-: 0.5$			$\mu: 1.75$	$\kappa: 0.25$
				$\Delta_M: 1.0$			$\omega: -10^{-4}$	$\phi: 0.12$
				$\Delta_m: 10^{-6}$			$\alpha: 0.0$	$\theta: 0.7$
				$\Delta_0: 0.1$				$\alpha: 0.8$
Epochs (Mean)	236	660	826	462	150	327	489	532
Epochs (StD)	60	125	212	103	57	98	181	93
Successes (%)	90	7	30	22	30	30	44	55

Table 3

Results in terms of number of epochs (mean value and standard deviation) and success rates for the parity-4 benchmark using the constrained learning algorithm of section 4 and other well known algorithms.

Table 3, the proposed method is compared with a host of other well known training algorithms. For the 4 bit parity problem, note that the successful negotiation of temporary minima by the proposed algorithm leads to a much higher success rate than all other methods. Moreover, learning is achieved in a relatively low average number of epochs (in this respect the proposed method is surpassed only by CG/PR, which, however, exhibits a very low success rate equal to just 30%).

The second benchmark is the "cancer3" classification problem of the PROBEN1 set [23]. It concerns the diagnosis of breast cancer, the task being to classify a tumor as benign or malignant based on cell descriptions gathered by a microscope. The problem has 9 real valued inputs, 2 binary outputs and consists of 699 examples, of which 350 are included in the training set. A network with 9 inputs, 5 hidden units and 2 outputs is employed here to solve this problem. For this benchmark, 10 learning trials were performed starting from different randomly chosen weights in the range -0.5 to 0.5. The maximum allowed number of epochs per trial was set to 5000. Comparative results of different algorithms concerning number of epochs and success rates for the "cancer3" benchmark are shown in Table 4. Note that the proposed algorithm is the only method fully successful in all trials with respect to Fahlman's criterion. Of all the other algorithms examined, only RPROP was able to solve the problem in just 20% of the trials performed.

	Proposed (sect. 4)	BPM OFF	BPM ON	RPROP	CG (PR)	CG (FR)	QP	DBD
	δP: 0.2 ξ: 0.5	NC	NC	η^+: 1.2 η^-: 0.5 Δ_M: 1.0 Δ_m: 10^{-6} Δ_0: 0.1			NC	NC
Epochs (Mean)	3272	F	F	346	F	F	F	F
Epochs (StD)	605	F	F	157	F	F	F	F
Successes (%)	100	0	0	20	0	0	0	0

Table 4

Results in terms of number of epochs (mean value and standard deviation) and success rates for the cancer benchmark using the constrained learning algorithm of section 4 and other well known algorithms.

5. Problem specific example: Polynomial factorization

Polynomial factorization is an important problem with applications in various areas of mathematics, mathematical physics and signal processing. It is a difficult problem for polynomials of more than one variable, where the fundamental theorem of Algebra is not applicable.

Consider, for example, a polynomial of two variables z_1 and z_2:

$$A(z_1, z_2) = \sum_{i=0}^{N_A} \sum_{j=0}^{N_A} a_{ij}\ z_1^i\ z_2^j \tag{44}$$

with N_A even, and $a_{00} = 1$. For the above polynomial, we seek to achieve an exact or approximate factorization of the form

$$A(z_1, z_2) \approx \prod_{i=1,2} A^{(i)}(z_1, z_2) \tag{45}$$

where

$$A^{(i)}(z_1, z_2) = \sum_{j=0}^{M_A} \sum_{k=0}^{M_A} v_{jk}^{(i)}\ z_1^j\ z_2^k \tag{46}$$

with $M_A = N_A/2$. We can try to find the coefficients $v_{jk}^{(i)}$ by considering P training patterns selected from the region $|z_1| < 1,\ |z_2| < 1$. The primary

purpose of the learning rule is thus to minimize with respect to the $v_{jk}^{(i)}$ a cost function of the form

$$E = \sum_p (\prod_{i=1,2} A^{(i)}(z_{1p}, z_{2p}) - A(z_{1p}, z_{2p}))^2 \tag{47}$$

Note that this cost function corresponds to a sigma-pi neural network with the elements of $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ as its synaptic weights. Unconstrained minimization of the cost function has been tried, but often leads to unsatisfactory results, because it can be easily trapped in flat minima. However, there is extra knowledge available for this problem, in the form of constraints between the coefficients of the desired factor polynomials and the coefficients of the original polynomial. More explicitly, if we assume that $A(z_1, z_2)$ is factorable, then these constraints can be expressed as follows:

$$\Phi^v_{j+(N_A+1)i} = a_{ij} - \sum_{l=1}^{i} \sum_{m=1}^{j} v_{lm}^{(1)} v_{i-l,j-m}^{(2)} = 0 \tag{48}$$

with $0 \leq i \leq N_A, 0 \leq j \leq N_A$. Thus, the objective of the adaptation process is to reach a minimum of the cost function of equation (47) with respect to the variables $v_{jk}^{(i)}$, which satisfies as best as possible the constraints $\boldsymbol{\Phi}^v = \mathbf{0}$, where $\boldsymbol{\Phi}^v = (\Phi^v_{j+(N_A+1)i}, 0 \leq i \leq N_A, 0 \leq j \leq N_A)$.

Here we incorporate the extra relations using our constrained optimization framework. Since the constraints have to be satisfied as best as possible upon termination of the training process, it is appropriate to utilize Case 1 of section 2. It turns out that the ensuing constrained learning algorithm can determine the factor polynomials in factorable cases and gives good approximate solutions in cases where the original polynomial is non-factorable.

Table 5 shows the coefficients of a factorable polynomial and the corresponding exact factor polynomials, as well as the solutions obtained by gradient descent with momentum and the constrained learning algorithm. Obviously, only the constrained learning algorithm results compare favorably with the exact result.

6. Conclusion

There are many types of *a priori* knowledge that can be incorporated into neural networks learning, in the form of additional relations that must be satisfied by the learning rule. These constraints are usually pointed out either by the selection of the network or learning rule itself, or from the specific problem at

	Exact	Constrained learning	Gradient descent
Product	$\begin{bmatrix} 1 & 1.5 & 0.5 \\ 4 & 5 & 2.25 \\ 3 & 6.5 & 1 \end{bmatrix}$	$\begin{bmatrix} 1.0000 & 1.5000 & 0.4997 \\ 4.0000 & 5.0038 & 2.2528 \\ 3.0020 & 6.5022 & 1.0092 \end{bmatrix}$	$\begin{bmatrix} 1.0000 & 1.2007 & 0.3604 \\ 0.8566 & 2.2905 & 1.0663 \\ 0.1834 & 0.7607 & 0.7888 \end{bmatrix}$
1st Factor	$\begin{bmatrix} 1 & 0.5 \\ 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 1.0000 & 0.4995 \\ 1.0010 & 1.9997 \end{bmatrix}$	$\begin{bmatrix} 1.0000 & 0.5996 \\ 0.4282 & 0.8874 \end{bmatrix}$
2nd Factor	$\begin{bmatrix} 1 & 1 \\ 3 & 0.5 \end{bmatrix}$	$\begin{bmatrix} 1.0000 & 1.0005 \\ 2.9990 & 0.5047 \end{bmatrix}$	$\begin{bmatrix} 1.0000 & 0.6010 \\ 0.4284 & 0.8889 \end{bmatrix}$

Table 5

Factorization results for a 2-D polynomial. The coefficient matrix $\boldsymbol{a}$ for the product polynomial, as well as the coefficient matrices $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ for the two factor polynomials are shown for the exact factorization, for the result obtained by the constrained optimization method described in section 5 and for the result obtained by the conventional method of BP.

hand which the neural network tries to learn. In this paper, we have derived a generic learning rule in which many types of additional knowledge, codified as mathematical relations satisfied by the synaptic weights, can be incorporated. We have also given specific examples of its application to neural network learning. The major benefit of this learning approach is to help relax the "black box" nature of artificial neural networks and combine the merits of both connectionist and knowledge based approaches for designing and implementing efficient information systems.

Appendix: Derivation of constrained learning algorithm

In this Appendix, evaluation of the Lagrange multipliers L_1, L_2 and $\boldsymbol{\Lambda}$ involved in the general constrained learning framework of section 2 is carried out.

By multiplying both sides of equation (12) by $\boldsymbol{G}^T$ and by taking into account equation (6) we obtain:

$$\delta Q = -\frac{L_1}{2L_2} I_{GG} - \frac{1}{2L_2} \boldsymbol{\Lambda}^T \boldsymbol{I_{GF}} \tag{49}$$

Solving for L_1 readily yields equation (21), which evaluates L_1 in terms of L_2 and $\mathbf{\Lambda}$.

By left multiplication of both sides of equation (12) by $\boldsymbol{F}$ and taking into account equations (7) and (21), we obtain

$$\kappa \mathbf{1} + \frac{\delta Q \boldsymbol{I_{GF}}}{I_{GG}} = \frac{\boldsymbol{R}\mathbf{\Lambda}}{2L_2} \tag{50}$$

where the matrix $\boldsymbol{R}$ is defined by equation (17). Solving equation (50) for $\mathbf{\Lambda}$ yields

$$\mathbf{\Lambda} = 2L_2 \kappa \boldsymbol{R}^{-1}\mathbf{1} + \frac{2L_2 \delta Q}{I_{GG}} \boldsymbol{R}^{-1}\boldsymbol{I_{GF}} \tag{51}$$

By substituting this equation into equation (13) we arrive at:

$$\kappa = -\frac{1 + \frac{2L_2 \delta Q}{I_{GG}} \mathbf{1}^T \boldsymbol{R}^{-1} \boldsymbol{I_{GF}}}{2L_2 (\boldsymbol{R}^{-1})_a} \tag{52}$$

We can now substitute this equation into equation (51) to obtain equation (20) which evaluates $\mathbf{\Lambda}$ in terms of L_2.

To evaluate L_2, we must substitute our expression for $d\boldsymbol{w}$ into equation (5). To make the algebra easier, we note that on account of equation (21), equation (12) can be written as:

$$d\boldsymbol{w} = \frac{\delta Q}{I_{GG}} \boldsymbol{G} + \frac{1}{2L_2} \boldsymbol{A} \tag{53}$$

where

$$\boldsymbol{A} = \frac{\mathbf{\Lambda}^T \boldsymbol{I_{GF}}}{I_{GG}} \boldsymbol{G} - \boldsymbol{F}^T \mathbf{\Lambda} \tag{54}$$

From the definition of $\boldsymbol{A}$ we can readily derive the following properties:

$$||\boldsymbol{A}||^2 = \mathbf{\Lambda}^T \boldsymbol{R} \mathbf{\Lambda}, \quad \boldsymbol{A}^T \boldsymbol{G} = 0 \tag{55}$$

Substituting equation (53) into equation (5) and taking into account equation (55), we can obtain a relation involving only L_2 and $\mathbf{\Lambda}$:

$$L_2 = -\frac{1}{2} \left[\frac{I_{GG} (\mathbf{\Lambda}^T \boldsymbol{R} \mathbf{\Lambda})}{I_{GG} (\delta P)^2 - (\delta Q)^2} \right]^{1/2} \tag{56}$$

where the negative square root sign has been selected on account of inequality (11).

By substituting equation (20) into equation (56) and solving for L_2, equation (19) is obtained, with Z given by equation (18). Evaluation of all Lagrange multipliers in terms of known quantities is now complete.

As a final note, let us discuss our choice for δQ. This choice is dictated by the demand that the quantity under the square root in equation (56) be positive. It can readily be seen by the first of equation (55) that $\mathbf{\Lambda}^T \boldsymbol{R} \mathbf{\Lambda} \geq 0$. Since $I_{GG} = \boldsymbol{G}^T \boldsymbol{G} \geq 0$, it follows from equation (56) that care must be taken to ensure that $I_{GG}(\delta P)^2 > (\delta Q)^2$. The simplest way to achieve this is to set $\delta Q = -\xi \delta P \sqrt{I_{GG}}$ with $0 < \xi < 1$.

References

[1] D. E. Rumelhart, J. E. Hinton and R. J. Williams, Learning internal representations by error propagation, in: *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. 1, *Foundations*, eds. D. E. Rumelhart and J. L. McLelland, MIT Press, 1986, pp. 318-362.

[2] D. Barber and D. Saad, Does extra knowledge necessarily improve generalization?, Neural Computation 8 (1996) 202-214.

[3] Y. le Cun, L. D. Jackel, B. E. Boser, J. S. Denker, H-P. Graf, I. Guyon, D. Henderson, R. E. Howard and W. Hubbard, Handwritten digit recognition: Applications of neural network chips and automatic learning, IEEE Communications Magazine (Nov. 1989) pp. 41–46.

[4] P. Simard, Y. le Cun and J. Denker, Efficient pattern recognition using a new transformation distance, in: *Advances in Neural Processing Systems,* eds. S. J. Hanson, J. D. Cowan and C. L. Giles, Morgan Kaufmann, 1993, pp. 50-58.

[5] S. Gold, A. Rangarajan and E. Mjolsness, Learning with preknowledge: clustering with point and graph matching distance, Neural Computation 8 (1996) 787-804.

[6] S. J. Perantonis and D. A. Karras, An efficient learning algorithm with momentum acceleration, Neural Networks 8 (1995) 237-249.

[7] R. Rohwer, The 'moving targets' training algorithm, in: *Advances in Neural Information Processing Systems,* ed. D. S. Touretzky, Morgan Kaufmann, 1990, pp. 558-565.

[8] T. Grossman, The CHIR algorithm for feed forward networks with binary weights, The 'moving targets' training algorithm, in: *Advances in Neural Information Processing Systems,* ed. D. S. Touretzky, Morgan Kaufmann, 1990, pp. 516-523.

[9] T. B. Trafalis and N. P Couellan, Neural network training via an affine scaling quadratic optimization algorithm, Neural Networks 9 (1996) 475-481.

[10] A. S. Weigend, D. E. Rumelhart and B. A. Huberman, Generalization by weight elimination with application to forecasting, in: *Advances in Neural Information Processing Systems,* ed. D. S. Touretzky, Morgan Kaufmann, 1991, pp. 875-882.

[11] A. Krogh, G. I. Thorbergsson and J. A. Hertz, A cost function for internal representations, in: *Advances in Neural Information Processing Systems,* ed. D. S. Touretzky, Morgan

Kaufmann, 1990, pp. 733-740.

[12] N. Ampazis, S. J. Perantonis and J. Taylor, Dynamics of multilayer networks in the vicinity of temporary minima, Neural Networks 12 (1999) 43-58.

[13] S. J. Perantonis, N. Ampazis, S. Varoufakis and G. Antoniou, Constrained learning in neural networks: Application to stable factorization of 2-D polynomials, Neural Proc. Lett. 7 (1998) 5-14.

[14] A. E. Bryson and W. F. Denham, A steepest ascent method for solving optimum programming problems, Journal App. Mech. 29 (1962) 247-257.

[15] S. S. Rao, *Optimization Theory and Applications*, New Delhi, Wiley Eastern, 1984.

[16] J. Hertz, A Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991.

[17] R. A. Jacobs, Increased rates of convergence through learning rate adaptation, Neural Networks 1 (1988) 295-307.

[18] S. E. Fahlman, Faster learning variations on back-propagation: An empirical study, in: *Proceedings of the Connectionist Models Summer School*, eds. D. Touretzky, G. Hinton and T. Sejnowski, Morgan Kaufmann, 1988, pp. 29-37.

[19] M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm. Proceedings of the International Conference on Neural Networks, San Francisco 1 (1993) 586-591.

[20] E. M. Johansson, F. U. Dowla and D. M. Goodman, Backpropagation learning for multilayer feedforward networks using the conjugate gradient method. International Journal of Neural Systems 2 (1992) 291-301.

[21] R. Yager and D. Filev, Generation of fuzzy rules by mountain clustering. Journal of Intelligent and Fuzzy Systems 2 (1994) 209-219.

[22] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, New York, Plenum, 1981.

[23] L. Prechelt, PROBEN1-A set of neural network benchmark problems and benchmarking rules, Technical Report 21/94, Universität Karlsruhe, Germany, 1994.