Optimization in an Error Backpropagation Neural Network Environment with a Performance Test on a Pattern Classification Problem

Fischer, Manfred M. and Staufer, Petra

Vienna University of Economics and Business, Vienna University of Economics and Business

1998

Manfred M. Fischer and Petra Staufer-Steinnocher

Optimization in an Error Backpropagation Neural Network Environment with a Performance Test on a Pattern Classification Problem

Paper

WSG Diskussionspapiere

WSG 62/98

Optimization in an Error Backpropagation Neural Network Environment with a Performance Test on a Pattern Classification Problem

Manfred M. Fischer and Petra Staufer

Institut für Wirtschafts- und Sozialgeographie

Wirtschaftsuniversität Wien

Department of Economic and Social Geography

Vienna University of Economics and Business Administration

WSG Discussion Papers

Abteilung für Theoretische und Angewandte Wirtschafts- und Sozialgeographie
Institut für Wirtschafts- und Sozialgeographie
Wirtschaftsuniversität Wien

Vorstand: o.Univ.Prof. Dr. Manfred M. Fischer
A - 1090 Wien, Augasse 2-6, Tel. ++43-(0)1-31336-4836

Redaktion: Univ.Ass. Dr. Petra Staufer

WSG 62/98

Optimization in an Error Backpropagation Neural Network Environment with a Performance Test on a Pattern Classification Problem

Manfred M. Fischer and Petra Staufer

WSG-Discussion Paper 62

March 1998

Gedruckt mit Unterstützung
des Bundesministerium
für Wissenschaft und Verkehr
in Wien

WSG Discussion Papers are interim
reports presenting work in progress
and papers which have been submitted
for publication elsewhere.

ISBN 3 85037 073 9

Optimization in an Error Backpropagation Neural Network Environment

with a Performance Test on a Real World Pattern Classification Problem

Manfred M. Fischer[a,b,*] Petra Staufer[a,*]

[a] Department of Economic and Social Geography, Wirtschaftsuniversität Wien
Augasse 2-6, A-1090 Vienna, Austria;
E-mail: {manfred.m.fischer, petra.staufer}@wu-wien.ac.at

[b] Institute for Urban and Regional Science, Austrian Academy of Sciences
Postgasse 7/4, A-1010 Vienna, Austria; E-mail: manfred.m.fischer@oeaw.ac.at

Abstract

Various techniques of optimizing the multiple class cross-entropy error function to train single hidden layer neural network classifiers with softmax output transfer functions are investigated on a real-world multispectral pixel-by-pixel classification problem that is of fundamental importance in remote sensing. These techniques include epoch-based and batch versions of backpropagation of gradient descent, PR-conjugate gradient and BFGS quasi-Newton errors. The method of choice depends upon the nature of the learning task and whether one wants to optimize learning for speed or generalization performance. It was found that, comparatively considered, gradient descent error backpropagation provided the best and most stable out-of-sample performance results across batch and epoch-based modes of operation. If the goal is to maximize learning speed and a sacrifice in generalisation is acceptable, then PR-conjugate gradient error backpropagation tends to be superior. If the training set is very large, stochastic epoch-based versions of local optimizers should be chosen utilizing a larger rather than a smaller epoch size to avoid inacceptable instabilities in the generalization results.

Keywords: Feedforward Neural Network Training, Numerical Optimization Techniques, Error Backpropagation, Cross-Entropy Error Function, Multispectral Pixel-by-Pixel Classification.

[*] *Acknowledgements:* The authors gratefully acknowledge the Department of Economic and Social Geography, Wirtschaftsuniversität Wien, for providing the software routines used in this study. The algorithms are based on the Numerical Recipes in C library [1] modified by *Adrian Trapletti* (supported by a grant of the Kuratorium of the Wirtschaftsuniversität Wien) as necessary to work in an error backpropagation neural network environment.

1 Introduction

In recent years there has been increasing use of neural networks for solving problems in the field of pattern classification (see [2], [3]). The main thrust of work in this area has been in the use of feedforward neural networks such as single hidden layer feedforward networks. Their popularity can be attributed largely to their general approximation capabilities. Inter alia, Hornik et al. [4] have demonstrated by rigorous mathematical proofs that such models can approximate virtually any function of interest to any desired degree of accuracy, provided sufficiently many hidden units are available. These results establish single hidden layer network models as a powerful class of universal approximators in general and pattern classifiers in particular. As such, failures in applications can be attributed to inadequate numbers of hidden units (i.e., inappropriate model choice) or inadequate learning (i.e., inappropriate parameter estimation). This contribution does not address the issue of model choice to attain a given degree of approximation, but focus is laid on the issue of network training (i.e., parameter estimation).

Several algorithms for adaptive training (learning) feedforward neural networks have recently been discovered. Many of them are based on the gradient descent technique. They generally depend on parameters that have to be specified by the user, as no theoretical basis exists for choosing them. The values of these parameters are often crucial for the success of the algorithms. One of the most widely — and often wildly — used is backpropagation of gradient descent errors, introduced and popularized by Rumelhart et al. [5], i.e., a combination of the backpropagation technique for calculating the partial derivatives of the error function and the gradient descent procedure for updating the network parameters.

Many enhancements of and variations to gradient descent backpropagation have been proposed (see, [6], [7]). These are mostly heuristic modifications with goals of increased speed of convergence, avoidance of local minima and/or improvement in the network model's ability to generalize, and usually evaluated on artificial benchmark problems. Another approach has been to draw upon a large body of theory in related fields such as statistical theory ([8]) and optimization theory ([9], [10]).

From the point of view of optimization theory network training is equivalent to minimizing an objective function that depends on the network parameters. This perspective opens the possibility to combine the backpropagation technique with more sophisticated optimization procedures for parameter adjustment, such as the conjugate gradient and the quasi-Newton procedure. No doubt, the performance of backpropagation training might be greatly influenced by the choice of the optimization procedure for parameter adaptation. This has been illustrated by a range of comparative studies, dominated by a focus on execution time and usually evaluated on artificial benchmark problems (see, for example, [11], [12], [13], [10], [14]).

The purpose of this contribution is to analyse the efficacy of backpropagation training with three optimization procedures for weight updating, the gradient descent (GD), the one-step Polak-Ribiere-conjugate gradient (PR-CG), the Broyden-Fletcher-Goldfarb-Shanno (BFGS) memoryless quasi-Newton algorithms, on a satellite image based pattern classification problem, with a challenging level of complexity and a larger size of training set. Two versions of off-line backpropagation training are considered: epoch-based learning (with epoch sizes $k = 30, 300, 600, 900$) and batch learning. They differ in how often the weights are updated. The batch version updates the weights after all patterns have been propagated through the network. The epoch-based version updates using in-

formation from K^* patterns randomly chosen from the training set. The evaluation is based on three performance indices, learning time (measured in both as the elapsed time in CPU-seconds and the number of iterations), required to convergence on a solution, out-of-sample classification performance (measured in terms of total classification accuracy) and stability (i.e., the ability of a network model with a given set of weights to converge, starting from different positions in parameter space).

The organization of the paper is as follows. In the next section a summarized description of single hidden layer networks as pattern classifiers is reviewed. Additionally, the network training problem is described as a problem of minimizing the multiple class cross-entropy error function. Training algorithms, whether used in off-line (epoch-based and batch) or on-line mode, are termed backpropagation algorithms if they use the technique of backpropagation for the evaluation of the error function derivatives (see Section 3) and some optimization scheme for parameter updating. They basically differ in the choice of the optimization procedure for weight adjustment. The three optimization procedures used in the simulations in this study are discussed in Section 4: the gradient descent, the PR-conjugate gradient and the BFGS quasi-Newton algorithms. These techniques are compared on a supervised multispectral pixel-by-pixel classification problem in which the classifier is trained with examples of the classes to be recognized in the data set. This is achieved by using limited ground survey information which specifies where examples of specific categories are to be found in the satellite imagery. The remote sensing pattern classification problem is characterized in section 5 along with a discussion of the relative strengths and weaknesses of the above optimization techniques when applied to solve the problem in batch and in epoch-based mode of operation. It will be demonstrated that the method of choice depends upon whether one wants to optimize learning speed or generalization performance. If the training set is very large, stochastic epoch-based rather than deterministic batch modes of operation tend to be preferable.

2 Single Hidden Layer Networks and the Network Training Problem

Suppose we are interested in approximating a classification function $\mathcal{F} : \mathcal{R}^N \mapsto \mathcal{R}^C$ which estimates the probability that a pattern belongs to one of a priori known mutually exclusive classes. The function $\mathcal{F}$ is not analytically known, but rather samples $S = \{s^1, \ldots, s^K\}$ with $s^k = (\mathbf{x}^k, \mathbf{y}^k)$ are generated by a process that is governed by $\mathcal{F}$, i.e., $\mathcal{F}(\mathbf{x}^k) = \mathbf{y}^k$. From the available samples we want to build a smooth approximation to $\mathcal{F}$. Note that in real-world applications, only a finite (i.e., small) number K of learning examples is available or can be used at the same time. Moreover, the samples contain noise.

To approximate $\mathcal{F}$ we consider the class of single hidden layer feedforward networks Φ, the leading case of neural network models. Φ consists of a combination of transfer functions $\varphi_j (j = 1, \ldots, J)$ and $\psi_c (c = 1, \ldots, C)$ that are represented by hidden units, and weighted forward connections between the input, hidden, and output units. The c-th output element of Φ is

$$\Phi(\mathbf{x}, \mathbf{w})_c = \psi_c \left(\sum_{j=0}^{J} w_{cj} \; \varphi_j \left(\sum_{n=0}^{N} w_{jn} \; \mathbf{x}_n \right) \right) \qquad 1 \leq c \leq C, \tag{1}$$

where N denotes the number of input units, J the number of hidden and C the number

of output elements (a priori given classes). $\mathbf{x} = (x_0, x_1, \ldots, x_N)$ is the input vector augmented with a bias signal x_0 that can be thought of as being generated by a 'dummy' unit (with index zero) whose output is clamped at 1. The w_{jn} represent input to hidden connection weights, and the w_{cj} hidden to output weights (including the biases). The symbol $\mathbf{w}$ is a convenient short-hand notion of the $\omega = [J(N+1) + C(J+1)]$-dimensional vector of all the w_{jn} and w_{cj} network weights and biases (i.e., the model parameters). $\varphi_j(\cdot)$ and $\psi_c(\cdot)$ are differentiable non-linear transfer (activation) functions of, respectively, the hidden units ($j = 1, \ldots, J$) and the output elements ($c = 1, \ldots, C$).

One of the major issues in neural network modelling includes the problem of selecting an appropriate member of model class Φ in view of a particular real-world application. This model specification problem involves both the choice of appropriate transfer functions $\varphi_j(\cdot)$ and $\psi_c(\cdot)$, and the determination of an adequate network topology of Φ. Clearly, the model choice problem and the network training problem (i.e., the determination of an optimal set of model parameters where optimality is defined in terms of an error function) are intertwined in the sense that if a good model specification can be found, the success of which depends on the particular problem, then the step of network training may become easier to perform.

In this contribution, however, we restrict our scope to the network training problem and to training algorithms that train a fixed (i.e., predetermined) member of class Φ. The approximation Φ to $\mathcal{F}$ then only depends on the learning samples, and the learning (training) algorithm that determines the parameters $\mathbf{w}$ from S and the model specification. Let us assume the hidden unit transfer functions $\varphi_j(\cdot)$ to be identical, $\varphi_j(\cdot) = \varphi(\cdot)$ for all $j = 1, \ldots, J$, and the logistic function. Thus, the output of the j-th hidden unit, denoted by z_j reads as

$$z_j = \varphi(net_j) = \frac{1}{1 + \exp(-net_j)} \qquad j = j, \ldots, J \tag{2}$$

with net_j representing the net input given by

$$net_j = \sum_{n=0}^{N} w_{jn}\ \mathbf{x}_n \qquad j = 1, \ldots, J. \tag{3}$$

Moreover, each unit c ($c = 1, \ldots, C$) of the output layer is assumed to have the same transfer function, denoted by $\psi(\cdot)$. Since the network should implement a pattern classifier with real valued outputs, a generalization of the logistic function known as softmax transfer function ([15]) represents an approriate choice. With this specification the c-th network output is

$$\Phi_c = \psi(net_c) = \frac{\exp(net_c)}{\sum_{c'=1}^{C} \exp(net_{c'})} \qquad c = 1, \ldots, C \tag{4}$$

with net_c representing the net input given by

$$net_c = \sum_{j=0}^{J} w_{cj} z_j = \sum_{j=0}^{J} w_{cj}\ \varphi(net_j) \qquad c = 1, \ldots, C. \tag{5}$$

The choice of this output transfer function is motivated by the goal of ensuring that the network outputs can be interpreted as probabilities of class membership, conditioned on the outputs z_j ($j = 1, \ldots, J$) of the hidden units (see [3, 238 pp.] for more details).

The process of determining optimal parameter values is called training or learning and may be formulated in terms of the minimization of an appropriate error function. The function that is minimized for the pattern classification problem of this study is the multiple class cross-entropy error function that is — following [3, p. 238] — defined (for batch learning) as a sum over all training patterns and all outputs as

$$\begin{aligned} E(\mathbf{w}) &= \sum_{k=1}^{K} E^k(\mathbf{w}) = -\sum_{k=1}^{K}\sum_{c=1}^{C} y_c^k \ \ln\left(\frac{\Phi(\mathbf{x}^k,\mathbf{w})_c}{y_c^k}\right) \\ &= -\sum_{k=1}^{K}\sum_{c=1}^{C} y_c^k \ \ln\left\{\frac{1}{y_c^k}\ \frac{\exp\left[\sum_j w_{cj}\left(1+\exp\left(-\sum_n w_{jn}\ x_n\right)\right)^{-1}\right]}{\sum_{c'}\exp\left[\sum_j w_{c'j}\left(1+\exp\left(-\sum_n w_{jn}\ x_n\right)\right)^{-1}\right]}\right\}. \end{aligned} \quad (6)$$

Φ_c represents the c-th component of the actual network output as a function of $\mathbf{x}^k$ and the weight vector $\mathbf{w}$, and may be interpreted as the network's estimate of the class membership probability. y_c^k is the target data which has a 1-of-C coding scheme so that $y_c^k = \delta_{cc'}$ for a training pattern $\mathbf{x}^k$ from class c' where $\delta_{cc'}$ denotes the Kronecker symbol with $\delta_{cc'} = 1$ for $c = c'$ and $\delta_{cc'} = 0$ otherwise. $E(\mathbf{w})$ can be calculated with one forward pass and the gradient $\nabla E(\mathbf{w})$ that is defined as

$$\nabla E(\mathbf{w}) = \left(\ldots, \sum_{k=1}^{K}\frac{\partial E^k(\mathbf{w})}{\partial w_{jn}}, \ldots, \sum_{k=1}^{K}\frac{\partial E^k(\mathbf{w})}{\partial w_{cj}}, \ldots\right) \quad (7)$$

where K is the number of patterns presented to the network model during training, and $E^k(\mathbf{w})$ the local cross-entropy error associated with pattern k. Optimal weights $\mathbf{w}^* = (\ldots, w_{jn}^*, \ldots, w_{cj}^*, \ldots)$ are obtained when $\Phi(\mathbf{x}^k)_c = \mathbf{y}_c^k$ for all $c = 1, \ldots, C$ and $k = 1, \ldots, K$. Characteristically, there exist many minima all of which satisfy

$$\nabla E(\mathbf{w}) = 0 \quad (8)$$

where $\nabla E(\mathbf{w})$ denotes the gradient of the error function in the ω-dimensional parameter space. The minimum for which the value of $E(\mathbf{w})$ is smallest is termed the global minimum while other minima are called local minima. But there is no guarantee about what kind of minimum is encountered. The problem is usually tackled by repeated application of the learning procedure from different starting configurations.

There are two basic approaches to find the minimum of the global error function E, off-line learning and on-line learning. They differ in how often the weights are updated. The *on-line* (i.e., pattern based) *learning* updates the weights after every single pattern s^k chosen at random from S, i.e., using only information from one pattern. In contrast, *off-line learning* updates the weights after K^* patterns randomly chosen from S have been propagated through the network, i.e., using information from K^* patterns in the training set. If $K^* = K$ off-line learning is known as *batch learning*, otherwise it is also termed *epoch-based learning* with an epoch size of K^* $(1 < K^* < K)$.

Both, the on-line and epoch-based (K^* small) versions are not consistent with optimization theory, but nevertheless have been found to be superior to batch learning on real world problems that show a realistic level of complexity and have a training set that goes beyond a critical threshold (see [16], [17]).

3 Optimization Strategy and the Backpropagation Technique

In the previous section the network training problem has been formulated as a problem of minimizing the multiple class cross-entropy error function, and, thus, as a special case of function approximation where no explicit model of the data is assumed. Most of the optimization procedures used to minimize functions are based on the same strategy. The minimization is a local iterative process in which an approximation to the function in a neighbourhood of the current point in parameter space is minimized. The approximation is often given by a first- or second-order Taylor expansion of the function. In the case of batch learning, the general scheme of the iteration process may be formulated as follows:

(i) choose an initial vector $\mathbf{w}$ in parameter space and set $\tau = 1$,

(ii) determine a search direction $\mathbf{d}(\tau)$ and a step size $\eta(\tau)$ so that

$$E(\mathbf{w}(\tau) + \eta(\tau)\, \mathbf{d}(\tau)) < E(\mathbf{w}(\tau)) \qquad \tau = 1, 2, \ldots \tag{9}$$

(iii) update the parameter vector

$$\mathbf{w}(\tau + 1) = \mathbf{w}(\tau) + \eta(\tau)\, \mathbf{d}(\tau) \qquad \tau = 1, 2, \ldots \tag{10}$$

(iv) if $dE(\mathbf{w})d\mathbf{w} \neq 0$ then set $\tau = \tau + 1$ and go to (ii), else return $\mathbf{w}(\tau + 1)$ as the desired minimum.

In the case of *on-line learning* the above scheme has to be slightly modified since this learning approach is based on the (local) error function E^k, and the parameter vector $\mathbf{w}^k(\tau)$ is updated after every presentation of $s^k = (\mathbf{x}^k, \mathbf{y}^k)$. In both cases, batch and on-line learning, determining the next current point in the iteration process is faced with two problems. *First,* the search direction $\mathbf{d}(\tau)$ has to be determined, i.e., in what direction in parameter space do we want to go in the search for a new current point. *Second,* once the search direction has been found, we have to decide how far to go in the specified direction, i.e., a step size $\eta(\tau)$ has to be determined. For solving these problems characteristically two types of operations have to be accomplished: *first,* the computation or the evaluation of the derivatives of the error function (6) with respect to the network parameters, and, *second*, the computation of the parameter $\eta(\tau)$ and the direction vector $\mathbf{d}(\tau)$ based upon these derivatives.

In the sequel we illustrate that the backpropagation technique is a powerful method for efficiently calculating the gradient of the cross-entropy error function (6) with respect to the parameter weights. Because the single hidden layer network is equivalent to a chain of function compositions (see equation (1)), the chain rule of differential calculus will play a major role in finding the gradient of function (6). In order to keep the notation uncluttered we will omit the superscript k (representing the k-th training pattern) from the terms, except the error function.

Let us consider, first, the partial derivatives of $E^k(\mathbf{w})$ with respect to the hidden-to-output connection parameters, i.e., the w_{cj} weights with $c = 1, \ldots, C$ and $j = 1, \ldots, J$.

Note that $E^k(\mathbf{w})$ depends on w_{cj} only via the summed input net_c to output unit c (cf. equation (5)). Thus, using the chain rule for differentiation one may express the partial derivatives of $E^k(\mathbf{w})$ we are looking for as (for training pattern k)

$$\frac{\partial E^k}{\partial w_{cj}} = \frac{\partial E^k}{\partial net_c} \; \frac{\partial net_c}{\partial w_{cj}}. \tag{11}$$

Recall from equation (5) that $net_c = \sum_{j=1}^{J} w_{cj} z_j$ where the sum is taken over the output of all hidden units $j = 1, \ldots, J$. Thus, the second term of the right hand side of equation (11) can be evaluated as follows:

$$\frac{\partial net_c}{\partial w_{cj}} = \frac{\partial}{\partial w_{cj}} \sum_{j=1}^{J} w_{cj} z_j = \frac{\partial}{\partial w_{cj}} \left[\sum_{j' \neq j} w_{cj'} z_{j'} + w_{cj} z_j \right] = z_j. \tag{12}$$

Substituting (12) into (11) we obtain

$$\frac{\partial E^k}{\partial w_{cj}} = \frac{\partial E^k}{\partial net_c} \; z_j \tag{13}$$

If we define the local error at the c-th node of the network, called delta, by

$$\delta_c := \frac{\partial E^k}{\partial net_c}, \tag{14}$$

we can express the partial derivatives of E with respect to w_{cj} as

$$\frac{\partial E^k}{\partial w_{cj}} = \delta_c \; z_j. \tag{15}$$

Equation (15) tells us that the required partial derivative $\partial E^k / \partial w_{cj}$ is obtained simply by multiplying the value of δ_c — associated with the output end of the connection parameter w_{cj} — with the value of z_j — associated with the input end of the connection parameter w_{cj}. Note that $z_0 = 1$. Thus, in order to evalute the partial derivatives in question we need only to calculate the value of δ_c for each $c = 1, \ldots, C$ and then apply (15).

This leads to the task to evaluate δ_c. Once again applying the chain rule we obtain

$$\delta_c = \frac{\partial E^k}{\partial net_c} = \sum_{c=1}^{C} \frac{\partial E^k}{\partial \Phi_{c'}} \; \frac{\partial \Phi_{c'}}{\partial net_c}. \tag{16}$$

From (6) we have

$$\frac{\partial E^k}{\partial \Phi_{c'}} = \frac{\partial}{\partial \Phi_{c'}} \left[-\sum_{c''=1}^{C} y_{c''} \; \ln \left[\frac{\Phi_{c''}}{y_{c''}} \right] \right] = -\frac{y_{c'}}{\Phi_{c'}} \tag{17}$$

and from (4)

$$\begin{aligned} \frac{\partial \Phi_{c'}}{\partial net_c} &= \frac{\partial}{\partial net_c} \left[\frac{\exp(net_c)}{\sum_{c'''} \exp(net_{c'''})} \right] \\ &= \delta_{cc'} \; \frac{\exp(net_c)}{\sum_{c'''} \exp(net_{c'''})} - \frac{\exp(net_{c'}) \exp(net_c)}{\left(\sum_{c'''} \exp(net_{c'''})\right)^2} = \delta_{cc'} \; \Phi_c - \Phi_c \; \Phi_{c'} \end{aligned} \tag{18}$$

where $\delta_{cc'}$ is the usual Kronecker symbol. Substituting (17) and (18) into (1) leads to

$$\begin{aligned}\delta_c &= \sum_{c'=1}^{C} \left[-\frac{y_{c'}}{\Phi_{c'}}\right] \; [\delta_{cc'} \; \Phi_c - \Phi_c \Phi_{c'}] = \sum_{c \neq c'} y_{c'} \; \Phi_c - y_c + y_c \; \Phi_c \\ &= \left[\sum_{c \neq c'} y_{c'} + y_c\right] \; \Phi_c - y_c = \Phi_c - y_c. \end{aligned} \tag{19}$$

Thus, the partial derivative we are looking for is

$$\frac{\partial E^k}{\partial w_{cj}} = \delta_c \; z_j = (\Phi_c - y_c) z_j. \tag{20}$$

Let us consider now the second set of partial derivatives, $\partial E^k / \partial w_{jn}$ for $j = 1, \ldots, J$ and $n = 1, \ldots, N$. This is a little more complicated. Again we apply the chain rule for differentiation and finally arrive at:

$$\frac{\partial E^k}{\partial w_{jn}} = \delta_j \; x_n \tag{21}$$

with

$$\delta_j = \varphi'(net_j) \sum_{c=1}^{C} w_{cj} \; \delta_c. \tag{22}$$

Note that the local error of the hidden units is determined on the basis of the local errors at the output layer (δ_c is given by (19)). The chain rule gives

$$\frac{\partial E^k}{\partial w_{jn}} = \frac{\partial E^k}{\partial z_j} \; \frac{\partial z_j}{\partial net_j} \; \frac{\partial net_j}{\partial w_{jn}} \tag{23}$$

with

$$\frac{\partial net_j}{\partial w_{jn}} = \frac{\partial}{\partial w_{jn}} \sum_{n=1}^{N} w_{jn} x_n = \frac{\partial}{\partial w_{jn}} \left[\sum_{n' \neq n} w_{jn'} x_{n'} + w_{jn} x_n\right] = x_n \tag{24}$$

$$\delta_j := \frac{\partial E^k}{\partial z_j} \; \frac{\partial z_j}{\partial net_j} \tag{25}$$

where the second term of the right hand side of equation (25) is evaluated as

$$\frac{\partial z_j}{\partial net_j} = \varphi'(net_j) = \varphi(net_j)(1 - \varphi(net_j)) \tag{26}$$

and the first term as

$$\begin{aligned}\frac{\partial E^k}{\partial z_j} &= \frac{\partial}{\partial z_j} \left\{ \sum_{c=1}^{C} y_c \; \ln \left[\frac{1}{y_c} \frac{\exp(\sum_j w_{cj} z_j)}{\sum_{c'} \exp(\sum_j w_{c'j} z_j)}\right] \right\} \\ &= \sum_{c=1}^{C} w_{cj} \, [\Phi_c - y_c] = \sum_{c=1}^{C} w_{cj} \; \delta_c. \end{aligned} \tag{27}$$

Substituting (26)-(27) into (25) leads to (22).

In summary, the backpropagation technique for evaluating the partial derivatives of the cross-entropy error function with respect to the w_{cj} and the w_{jn} parameters can be described with three major steps:

Step 1: Feedforward Computation: Select (preferably at random) a training pattern $(\mathbf{x}^k, \mathbf{y}^k)$ from the set of training samples and propagate $\mathbf{x}^k$ forward through the network using equations (2)–(5), thus generating the hidden and output unit activations based on current weight settings.

Step 2: Set of Partial Derivatives $\partial E^k/\partial w_{cj}$: Compute the δ_c for all the output units $c = 1, \ldots, C$ using (19), and utilize (20) to evaluate the required derivatives.

Step 3: Set of Partial Derivatives $\partial E^k/\partial w_{jn}$: Backpropagate the deltas using (22) and (19) backward to the hidden layer to obtain δ_j for each hidden unit $j = 1, \ldots, J$ in the network model, and utilize (21) to evaluate the required derivatives.

In *on-line learning* the error function gradient is evaluated for just one pattern at a time, and the parameters updated using (11) where the different patterns $s^k = (\mathbf{x}^k, \mathbf{y}^k)$ in the training set S can be considered in sequence, or more typically selected at random. For *off-line* (i.e., batch and epoch-based) *learning*, the derivative of the total error $E(\mathbf{w})$ can be obtained by repeating the above three steps for each training sample s^k, and then summing over all samples, i.e.,

$$\frac{\partial E}{\partial w_{cj}} = \sum_{k=1}^{K^*} \frac{\partial E^k}{\partial w_{cj}} \tag{28}$$

and

$$\frac{\partial E}{\partial w_{jn}} = \sum_{k=1}^{K^*} \frac{\partial E^k}{\partial w_{jn}} \tag{29}$$

with $K^* = K$ in the case of batch learning, and $K \gg K^* > 1$ in the case of epoch-based learning with epoch size K^*. It is worthwhile to note that in the stochastic version of epoch-based learning K^* training patterns are chosen randomly in each iteration step of the optimization strategy.

4 Optimization Techniques for Parameter Adjustments

In numerical optimization different techniques for the computation of the parameter $\eta(\tau)$ and the direction vector $\mathbf{d}(\tau)$ are known (see, e.g., [18], [19]). We consider three techniques that are used for the pattern classification task to be described in Section 5:

(i) The *steepest-descent (gradient) method (GD)* defines the direction as the negative gradient

$$\mathbf{d}(\tau) := -\nabla E(\mathbf{w}(\tau))\,. \tag{30}$$

(ii) *Conjugate gradient (CG) methods* calculate the actual search direction $\mathbf{d}(\tau)$ as a linear combination of the gradient vector and the previous search directions (see [20]). In the PR-CG algorithm, the Polak-Ribiere variant of conjugate gradient procedures (see [1]), that is used in the simulations presented in Section 5, the search direction is computed as

$$\mathbf{d}(\tau) := -\nabla E(\mathbf{w}(\tau)) + \beta(\tau)\ \mathbf{d}(\tau - 1) \qquad \tau = 1, 2, \ldots \tag{31}$$

with

$$\mathbf{d}(0) = -\nabla E(\mathbf{w}(0)) \tag{32}$$

where $\beta(\tau)$ is a scalar parameter that ensures that the sequence of vectors $\mathbf{d}(\tau)$ satisfying the following condition expressed as

$$\beta(\tau) = \frac{[\nabla E(\mathbf{w}(\tau)) - \nabla E(\mathbf{w}(\tau - 1))]^T\ \nabla E(\mathbf{w}(\tau))}{\nabla E(\mathbf{w}(\tau - 1))^T\ \nabla E(\mathbf{w}(\tau - 1))}. \tag{33}$$

$\mathbf{w}(\tau - 1)^T$ is the transpose of $\mathbf{w}(\tau - 1)$. Note that the CG algorithm utilizes information about the direction search $\mathbf{d}(\tau - 1)$ from the previous iteration in order to accelerate convergence, and each search direction would be conjugate if the objective function would be quadratic.

(iii) *Quasi-Newton* — also called variable metric — *methods* employ the differences of two successive iteration points τ and $\tau + 1$, and the difference of the corresponding gradients to approximate the inverse Hessian matrix. The most commonly used update technique is the BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm (see [18], [19], [1]) that determines the search direction as

$$\mathbf{d}(\tau) = -\mathbf{H}(\tau)\ \nabla E(\mathbf{w}(\tau)) \tag{34}$$

where $\mathbf{H}(\tau)$ is some $\omega \times \omega$ symmetric positive definite matrix and denotes the current approximation to the inverse of the Hessian matrix, i.e.,

$$\mathbf{H}(\tau) \cong \left[\nabla^2\ E(\mathbf{w}(\tau))\right]^{-1} \tag{35}$$

where

$$\begin{aligned} \mathbf{H}(\tau) \quad &= \quad \left\{\mathbf{I} - \frac{\mathbf{d}(\tau - 1)\,(\mathbf{g}(\tau - 1))^T}{(\mathbf{d}(\tau - 1))^T\,\mathbf{g}(\tau - 1)}\right\} \mathbf{H}(\tau - 1) \left\{\mathbf{I} - \frac{\mathbf{g}(\tau - 1)\,(\mathbf{d}(\tau - 1))^T}{(\mathbf{d}(\tau - 1))^T\,\mathbf{g}(\tau - 1)}\right\} \\ &+ \quad \frac{\mathbf{d}(\tau - 1)\,(\mathbf{d}(\tau - 1))^T}{(\mathbf{d}(\tau - 1))^T\,\mathbf{d}(\tau - 1)} \end{aligned} \tag{36}$$

with

$$\mathbf{g}(\tau - 1) := \nabla E(\mathbf{w}(\tau)) - \nabla E(\mathbf{w}(\tau - 1)). \tag{37}$$

$\mathbf{H}$ is initialised usually with the identity matrix $\mathbf{I}$ and updated at each iteration using only gradient differences to approximate second order information. The inverse Hessian is more closely approximated as iterations proceed.

Both the PR-CG and the BFGS algorithms raise the calculation complexity per training iteration considerably since they have to perform a one-dimensional linear search in order to determine an appropriate step size. A line search involves several calculations of either the global error function E or its derivative, both of which raise the complexity. Characteristically, the parameter $\eta = \eta(\tau)$ is chosen to minimize

$$E(\tau) = E(\mathbf{w}(\tau)) + \eta\ \mathbf{d}(\tau) \tag{38}$$

in the τ-th iteration. This gives an automatic procedure for setting the step length, once the search direction $\mathbf{d}(\tau)$ has been determined.

All these three procedures use only first-order derivative information of the error function. The derivatives can, thus, be calculated efficiently by backpropagation as shown in Section 3. The steepest descent algorithm has the great advantage of being very simple and cheap to implement. One of its limitations is the need to choose a suitable step size η by trial and error. Inefficiency is primarily due to the fact that the minimization directions and step sizes may be poorly chosen. Unless the first step is chosen such that it leads directly to the minimum, steepest descent will zig-zag with many small steps.

In contrast, the conjugate gradient and quasi-Newton procedures are intrinsically off-line parameter adjustment techniques, and evidently more sophisticated optimization procedures. In terms of complexity and convergence property, the conjugate gradient can be regarded as being somewhat intermediate between the method of gradient descent and the quasi-Newton technique ([21]). Its advantage is the simplicity for estimating optimal values of the coefficients $\eta(\tau)$ and $\beta(\tau)$ at each iteration. No $\omega \times \omega$-dimensional matrices $\mathbf{H}(\tau)$ need to be generated as in the quasi-Newton procedures. The search direction is chosen by appropriately setting the β so that $\mathbf{d}$ distorts as little as possible the minimization achieved by the previous search step. A major difficulty is that for the non-quadratic error function (6) the obtained directions are *not* necessarily descent directions and numerical instability can result ([22]). Periodically, it might be necessary to restart the optimization process by a search in the steepest descent direction when a non-descent search direction is generated. It is worthwhile to mention that the gradient descent procedures can be viewed as a form of gradient descent with an adaptive momentum $\beta(\tau)$, the important difference being that $\eta(\tau)$ and $\beta(\tau)$ in conjugate gradient are automatically determined at each iteration (see equations (33) and (38)).

But the conjugate gradient methods are not as effective as some quasi-Newton procedures. They require approximately twice as many gradient evaluations as the quasi-Newton methods. However, they save time and memory required for calculating the $\omega \times \omega$-dimensional matrices $\mathbf{H}(\tau)$, especially in the case of large-sized classification problems ([23]). The quasi-Newton methods provide many advantages of the Newton method while using only first-order information about the objective function (6). The matrices $\mathbf{H}(\tau)$ are positive definite approximations of the inverse Hessian matrix obtained from gradient information. Thus, it is not required to evaluate second-order derivatives of E. The algorithms such as BFGS are always stable since $\mathbf{d}(\tau)$ is always a descent search direction. They are today the most efficient and sophisticated optimization techniques for batch training. But they are expensive both in computation and memory. Large-sized real-world classification problems implying larger ω could lead to prohibitive memory requirements ([23]).

5 The Pattern Classification Task and Test Results

The following training procedures that represent the major classes of the optimization techniques are compared:

(i) GD: error backpropagation with gradient descent minimization and with fixed and constant step sizes (i.e., the standard backpropagation technique),

(ii) PR-CG: error backpropagation with the Polak-Ribiere conjugate gradient minimization,

(iii) BFGS: error backpropagation with the Broyden-Fletcher-Goldfarb-Shanno quasi-Newton minimization

in batch mode as well as in epoch-based operation with epoch sizes $K^* = 30, 300, 600, 900$.

These procedures are compared on a supervised multispectral pixel-by-pixel classification problem in which the classifier is trained with examples of the classes to be recognized in the data set. This is achieved by using limited ground survey information which specifies where examples of specific categories are to be found in the satellite imagery. Such ground truth information has been gathered on sites which are well representative of the larger area analyzed from space. The image data set consists of 2,460 pixels (resolution cells) selected from a Landsat Thematic Mapper (TM) scene (270×360 pixels) from the city of Vienna and its northern surroundings (observation date: June 5, 1985, location of the centre: $16°23'E$, $48°14'N$; TM Quarter Scene 190-026/4). The six Landsat TM spectral bands used are blue (SB1), green (SB2), red (SB3), near infrared (SB4), and mid infrared (SB5 and SB7), excluding the thermal band with only a 120 meter ground resolution. Thus, each TM pixel represents a ground area of 30m×30m and has six spectral band values varing over 256 digital numbers (16 bits).

The purpose of the multispectral classification task at hand is to distinguish between the eight classes of urban land use listed in Table 1. The classes chosen are meaningful to photointerpreters and land use managers, but are not necessarily spectrally homogeneous. This classification problem used to evaluate the performance of the above training procedures in a real-world context, is challenging. The pixel-based remotely sensed spectral band values are noisy and sometimes unreliable. Some of the urban land use classes are sparsely distributed in the image. The number of training sites is small relative to the number of land use categories (one site training case). The training sites vary between 154 pixels (class suburban) and 602 pixels (class woodland and public gardens with trees). The above mentioned six TM bands provide the data set input for each pixel, with values scaled to the interval $[0.1, 0.9]$. This approach resulted in a database consisting of 2,460 pixels (about 2.5 percent of all the pixels in the scene) that are described by six-dimensional feature vectors, each tagged with its correct class membership. The set was divided into a training set (two thirds of the training site pixels) and a testing set by stratified random sampling — stratified in terms of the eight classes. Pixels from the testing set are not used during network training and serve only to evaluate out-of-sample (generalization) performance accuracy (measured in terms of total classification accuracy) when the trained classifier is presented with novel data. The goal is to predict the correct class category for the test sample of pixels. In remote sensing classification tasks generalization performance can be more important than fast learning.

Table 1 to be placed about here

A single hidden layer network classifier, with $N = 6$ input units (representing the six-dimensional feature vectors), $J = 14$ hidden units and $C = 8$ output units (representing the eight urban land use classes), was used in the study along with logistic hidden and softmax output transfer functions. Using a pruning technique $J = 14$ has been found in a previous study ([24]) to control the effective complexity of the network (complexity in terms of the model parameters). The network was initialized with random weights in the range $[-0.1, 0.1]$. Weights were updated in batch and epoch-based modes. The latter with a range of four epoch sizes ($K^* = 30, 300, 600, 900$). Learning was stopped when every element of the gradient vector had an absolute value of less than 10^{-6}. This termination criterion is considered as a realistic test for convergence on a minimum ([25]). Otherwise, training was terminated if the number of iterations exceeded 100,000. The test set was presented at convergence or after a maximum of 100,000 iterations in order to monitor generalization.

Each experiment (i.e., combination of training algorithm with tuning parameters) was repeated 10 times, the network classifier being initialized with a different set of random weights before each trial. To enable more accurate comparisons, the classifier was initialised with the same 10 sets of random weights for all experiments. All experiments were done on a SUN Ultra 1 workstation. For implementation of PR-CG and BFGS, we used library algorithms [1] modified as necessary to work in an error backpropagation neural network environment.

Tables 2 and 3 to be placed about here

Table 2 presents the results for experiments involving batch mode of operation. Averages and standard deviations of performance indices were calculated only for those trials which converged within the 100,000 iterations limit. The times shown are CPU seconds in the sense that they exclude overheads such as scoring on the test set and screen display of progress information. In-sample (out-of-sample) performance is measured in terms of the percentage of training (testing) pixels correctly classified at convergence. In order to do justice to each algorithm, optimal combinations of parameters were systematically sought. GD requires time consuming tuning of the learning rate parameter η to get optimum performance. The possible restarts and line search combinations for PR-CG and BFGS also require tuning, particularly when using inexact searches. Since the error surface is often highly non-quadratic, it is important to use a line search that deals successfully with non-convexities etc.

Figures 1 and 2 to be placed about here

Although GD could quickly learn the training set, it could not find a local minimum in less than 6,499 iterations. Convergence to a minimum was impossible except with very low η-values which make convergence extremely slow. By contrast, PR-CG and BFGS proved to be much faster on average. This can also be observed from the learning curves displayed in Figure 1. To avoid cluttering the graphs Figure 1 shows only the learning curves averaged over all the converged simulations of the 10 trials. The GD-learning curve clearly illustrates the extremely slow convergence of GD-optimization. In contrast, PR-

CG and BFGS have the advantage of faster convergence. The rate of convergence for the PR-CG is significantly higher than that of BFGS. But the difference is not statistically significant. The higher convergence rate of both these techniques was offset by greater computational complexity in terms of CPU-time (see Table 2) and exhaustive memory requirements. In contrast to BFGS, PR-CG saves time and memory needed for computing a totally dense matrix $\mathbf{H}(\tau)$ at each iteration step τ, but requires approximately twice as many gradient evaluations.

The average values of the multiple class cross-entropy function do seem to indicate that PR-CG tended to find better local minima tan any other procedure, and this conclusion is corroborated by the fact that the standard deviation after training in the ten runs is significantly lower as shown in Table 2. Especially, BFGS appears to be more prone to fall into local minima as indicated from the rather hight standard deviation. Moreover, Table 2 clearly indicates that better generalization performance is not the result of finding a lower minimum (see also [22]). PR-CG and BFGS seem to utilize information to modify the direction of steepest descent that resulted in significantly poorer generalization on our task. In fact, GD outperforms PR-CG by 9.60 percentage points and BFGS by 8.10 percentage points on average. An interesting conclusion from this comparative study is that generalization performance can vary between algorithms and even between different trials of the same algorithm, despite all of them finding a local minimum. It is important to note that GD-generalization performance varies between 80.49 and 86.22 percent of classification accuracy, while PR-CG generalization performance varies between 70.21 and 78.54, and BFGS between 67.80 and 79.88 percent only.

The second series of experiments involves epoch-based rather than batch learning, with a range of epoch sizes $K^* = 900, 600, 300$ and 30. The results obtained are summarized in Table 3 along with the corresponding learning curves displayed in Figure 2. Epoch-based learning strategies may be more effective than batch learning, expecially when the number K of training examples is very large and many training examples possess redundant information in the sense that many contributions to the gradient are very similar. Epoch-based updating makes the search path in the parameter space stochastic when the input vector is drawn at random. The main difficulty with stochastic epoch-based learning is its apparent inability to converge on a minimum within the 100,000 iterations limit.

While modifications to increase speed are important and attract most attention, generalization ability is perhaps more important in applications such as pixel-by-pixel classification of remotely sensed data. Important differences in generalization performance between batch and epoch-based learning may be observed. *First*, with larger K^* epoch based learning tends to outperform batch out-of-sample performance. This is especially true for PR-CG optimization. The best generalization results for all three optimization techniques are obtained for $K^* = 900$. The generalization ability of PR-CG is improved by 8.92 percentage points in classification accuracy on average (at the expense of less stable solutions), and that of BFGS by 6.74 percentage points. GD produces slightly better and much more stable generalization results. *Second*, better generalization is not the result of finding lower multiple class cross-entropy function values. *Third*, out-of-sample performance tends to slightly decrease with decreasing epoch size and at the same time the uncertainty in the generalization obtained increases. This can be seen from the larger amplitude of the oscillation in Figure 2(c) and especially Figure 2(d). Oscillation in results cause a larger standard deviation.

6 Concluding Remarks

Three techniques of optimizing the multiple class cross-entropy error function to train single hidden layer neural network classifiers with softmax output transfer functions were analysed on a real-world multispectral pixel-by-pixel classification problem that is of fundamental importance in remote sensing. These techniques (GD, PR-CG and BFGS) are but a small sample of all the techniques that have been studied in error backpropagation neural network environments. But the results of previous studies [9; 11; 17; 22] indicate that these techniques are representative of the most important approaches. Thus, the results of Section 5 can provide interesting empirical evidence in view of a challenging real-world pattern classification problem using remotely sensed data. Much contradictory information exists on the relative merits and demerits of different training strategies, especially with respect to their training speeds. We did not attempt to resolve these issues, but we focused on providing some evidence on how the strategies (batch versus epoch-based operation) and techniques (GD, PR-CG and BFGS) differ with respect to their training speed and their ability to generalize.

The choice of optimization strategy (batch versus epoch-based mode of operation) and of the optimization technique (GD, PR-CG and BFGS) depends on the nature of the learning task and whether one wants to optimize learning for speed or generalization performance. If the goal is to maximize learning speed on a pattern classification problem and a sacrifice in generalization is acceptable, then PR-CG error backpropagation, the most mature technique, would be the method of choice. Where high generalization and stability is more important than faster learning, then GD error backpropagation exhibits superiority over PR-CG and BFGS in view of our pixel-by-pixel pattern classification task — independently of the mode of operation — but requires time consuming tuning of the learning parameter η to achieve "best" out-of-sample performance. If the training set is very large, stochastic epoch-based rather than deterministic batch modes of operation should be chosen, with a larger rather than a smaller epoch size. Much work on such optimizers, however, is still required before they can be utilized with the same confidence and ease that batch local optimizers one currently used.

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge, MA: Cambridge University Press, 2 ed., 1992.

[2] Y.-H. Pao, *Adaptive Pattern Recognition and Neural Networks*. Reading, MA: Addison Wesley, 1989.

[3] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 4, pp. 359–366, 1989.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart, L. J. McClelland, and the PDP Research Group, eds.), vol. 1, pp. 318–332, Cambridge, MA: MIT Press, 1986.

[6] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–307, 1988.

[7] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the convergence of rhe back-propagation method," *Biological Cybernetics*, vol. 59, pp. 257–263, 1988.

[8] H. White, "Some asymptotic results for learning in single hidden-layer feedforward networks," *Journal of the American Statistical Association*, vol. 84, pp. 1003–1013, 1989.

[9] R. Battiti, "First- and second order methods for learning: Between steepest descent and Newton's method," *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.

[10] P. P. Van der Smagt, "Minimization methods for training feedforward neural networks," *Neural Networks*, vol. 7, no. 1, pp. 1–11, 1994.

[11] E. Barnard, "Optimization for training neural nets," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 232–240, 1992.

[12] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.

[13] R. Brunelli, "Training neural nets through stochastic minimization," *Neural Networks*, vol. 7, no. 9, pp. 1405–1412, 1994.

[14] C. de Groot and D. Würtz, "'Plain backpropagation' and advanced optimization algorithms: A comparative study," *Neurocomputing*, vol. 6, no. 2, pp. 153–161, 1994.

[15] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing: Algorithms, Architectures and Applications* (F. Fogelman Soulié and J. Hérault, eds.), pp. 227–236, New York: Springer, 1990.

[16] Y. Le Cun, "Generalization and network design strategies," in *Connections in Perspective* (M. Pfeifer, ed.), pp. 143–155, Amsterdam: North-Holland, 1989.

[17] W. Schiffmann, M. Jost, and R. Werner, "Comparison of optimized backpropagation algorithms," in *European Symposium on Artificial Neural Networks* (M. Verleysen, ed.), (Brussels), pp. 97–104, 1993.

[18] P. Luenberger, *Linear and Nonlinear Programing*. Reading, MA: Addison-Wesley, 2 ed., 1984.

[19] R. Fletcher, *Practical Methods of Optimization*. New York: Wiley-Interscience, 1986.

[20] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[21] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*. Chichester New York Brisbane Toronto Singapore: Wiley, 1993.

[22] R. Battiti and G. Tecchiolli, "Learning with first, second and no derivatives: A case study in high energy physics," *Neurocomputing*, vol. 6, pp. 181–2061, 1994.

[23] D. F. Shanno, "Recent advances in numerical techniques for large-scale optimization," in *Neural Networks for Robotics and Control* (W. T. Miller, ed.), (Cambridge, MA), pp. 171–178, MIT Press, 1990.

[24] M. M. Fischer, S. Gopal, P. Staufer, and K. Steinnocher, "Evaluation of neural pattern classifiers for a remote sensing application," *Geographical Systems*, vol. 4, no. 2, pp. 195–224 and 243–244, 1997.

[25] D. F. Shanno, "Conjugate gradient methods with inexact searches," *Mathematics of Operations Research*, vol. 3, no. 3, pp. 244–256, 1978.

Table 1: Classes and Number of Training/Testing Pixels

		Pixels	
	Description	Training	Testing
Class c_1	Mixed grass and arable farmland	167	83
Class c_2	Vineyards & areas with low vegetation cover	285	142
Class c_3	Asphalt & concrete surfaces	128	64
Class c_4	Woodland & public gardens with trees	402	200
Class c_5	Low density suburban areas	102	52
Class c_6	Densely built up urban areas	296	148
Class c_7	Water courses	153	77
Class c_8	Stagnant water bodies	107	54
Total Number of Pixels		1,640	820

Table 2: Batch learning: Comparative performance of error backpropagation with different optimization techniques*

	Error Backpropagation with		
	GD ($\eta = 0.0008$)	**PR-CG**	**BFGS**
Number Converged	9	10	8
Time	0.43 (0.67)	9.36 (12.49)	48.85 (44.00)
Iterations	19,443 (25,343.86)	8,306 (4,650.37)	9,582.75 (16,659.89)
Function Values	263.35 (29.63)	110.271 (17.15)	244.00 (79.58)
In-Sample Classification Accuracy	93.08 (0.82)	96.65 (0.42)	93.81 (1.63)
Out-of-Sample Classification Accuracy	84.20 (1.83)	74.60 (2.26)	76.10 (3.77)

* *Performance values represent the mean (standard deviation in brackets) of converged simulations.* *Number Converged*: number of simulations converged within 100,000 iterations, out of 10 trials differing in the initial random weights. The same 10 sets of weights were used for each algorithm. *Time*: CPU seconds required to reach the convergence condition. *Function Value*: multiple class cross-entropy error function value at convergence *In-Sample-Classification Accuracy*: percentage of training pixels correctly classified at convergence. *Out-of-Sample-Classification Accuracy*: percentage of test pixels correctly classified at convergence.

Table 3: Epoch based learning: Comparative performance of error backpropagation with different optimization techniques*

	Error Backpropagation with		
	GD	**PR-CG**	**BFGS**
Epoch Size $K^* = 900$ (GD: $\eta = 0.0003$)			
Time	21.97 (3.44)	28.14 (4.67)	42.74 (6.84)
Function Values	249.01 (3.24)	210.16 (16.06)	196.05 (21.23)
In-Sample Classification Accuracy	93.43 (0.17)	94.18 (0.59)	94.74 (0.82)
Out-of-Sample Classification Accuracy	85.67 (0.73)	83.52 (4.38)	82.84 (3.85)
Epoch Size $K^* = 600$ (GD: $\eta = 0.0001$)			
Time	21.88 (2.57)	23.45 (3.14)	37.91 (5.39)
Function Values	312.76 (3.82)	167.05 (20.53)	151.86 (17.04)
In-Sample Classification Accuracy	92.27 (0.08)	95.16 (0.59)	95.74 (0.53)
Out-of-Sample Classification Accuracy	85.28 (1.07)	79.60 (5.42)	76.14 (4.92)
Epoch Size $K^* = 300$ (GD: $\eta = 0.008$)			
Time	20.82 (1.98)	17.22 (4.33)	30.33 (4.95)
Function Values	196.33 (7.67)	157.38 (11.68)	190.37 (29.34)
In-Sample Classification Accuracy	94.87 (0.28)	95.41 (0.55)	95.38 (0.39)
Out-of-Sample Classification Accuracy	83.56 (2.41)	77.50 (4.12)	76.12 (3.73)
Epoch Size $K^* = 30$ (GD: $\eta = 0.08$)			
Time	17.10 (0.45)	11.37 (0.65)	20.24 (3.34)
Function Values	247.58 (27.80)	790.47 (310.90)	1,451.83 (374.08)
In-Sample Classification Accuracy	93.59 (1.20)	84.05 (9.14)	81.66 (5.02)
Out-of-Sample Classification Accuracy	81.29 (3.17)	74.65 (6.84)	75.07 (4.40)

* *Performance values represent the mean (standard deviation in brackets) of 10 simulations* differing in the initial random weights. *Time*: CPU seconds required to reach the convergence condition. *Function Value*: multiple class cross-entropy function value after 10^5 iterations. *In-Sample-Classification Accuracy*: percentage of training pixels correctly classified after 10^5 iterations. *Out-of-Sample-Classification Accuracy*: percentage of test pixels correctly classified after 10^5 iterations.

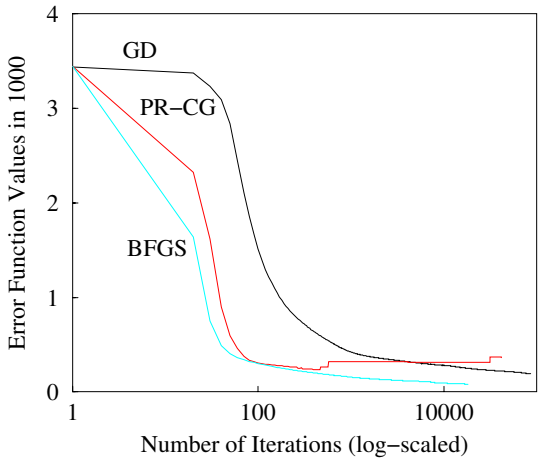

Figure 1: Batch learning curves as a function of training time: The effects of different optimization techniques [averaged value of converged simulations]

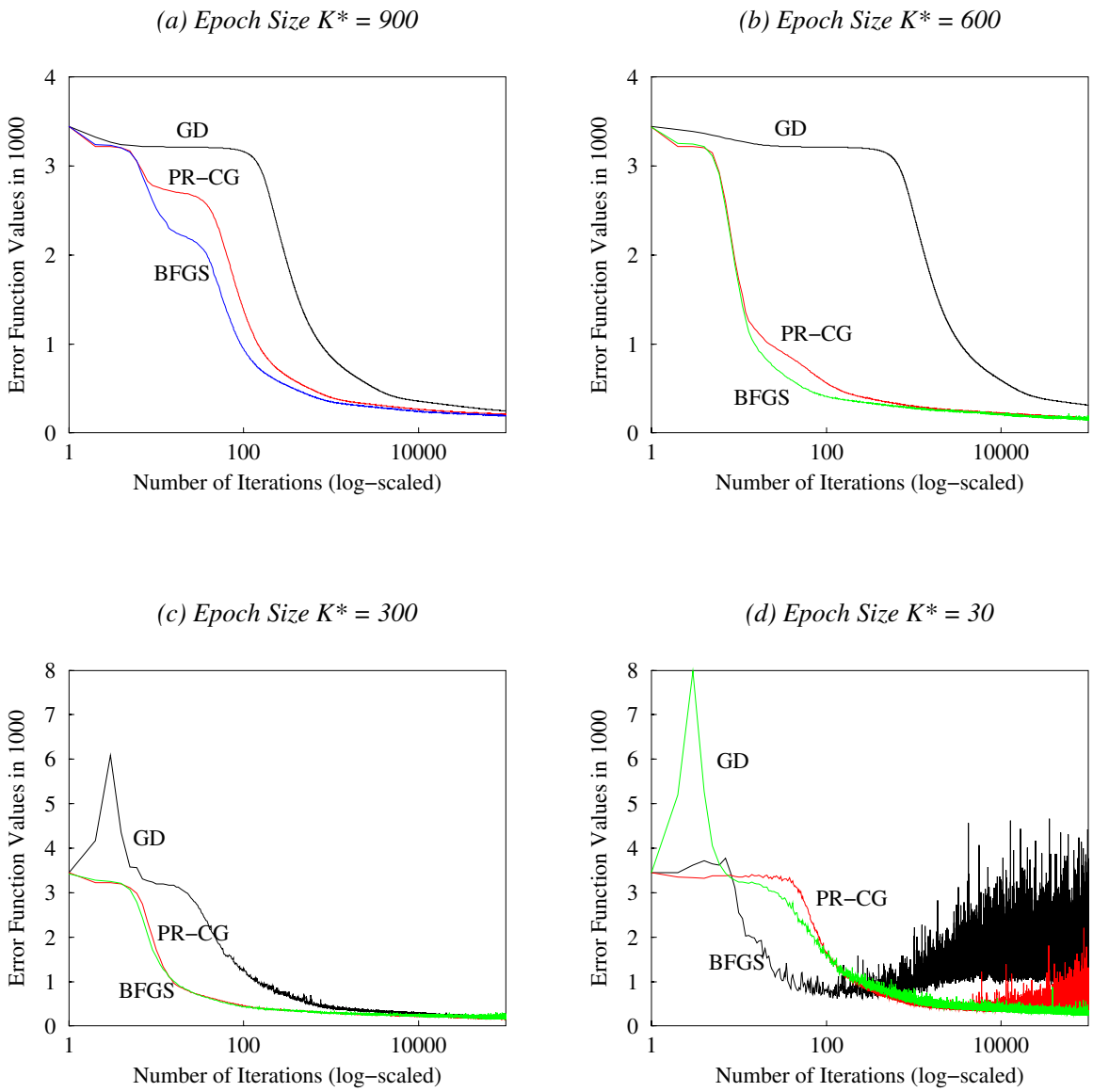

Figure 2: Epoch−based learning curves as a function of training time: The effects of different optimization techniques [averaged values over 10 trials]

List of Symbols Used

n	input unit label
j	hidden unit label
c	output unit (class) label
N	number of input units
J	number of hidden units
C	number of output units (classes)
K	number of learning patterns
K^*	epoch size
k	learning pattern label
$\mathbf{x}^k$	k-th learning input pattern
$\mathbf{y}^k$	k-th learning output pattern
ψ_c	transfer function of the c-th output unit
φ_j	transfer function of the j-th hidden unit
φ'	derivative of φ
exp	exponential function
ln	logarithm to base e
$\mathcal{R}$	space of real numbers
x^T	transpose of x
$\delta_{cc'}$	Kronecker symbol
$\mathbf{w}$	vector of all w_{cj} and w_{jn} network weights and biases
$\mathbf{w}^*$	vector of optimal w^*_{cj}- and w^*_{jn}-patterns
w_{cj}	connection weight from hidden unit j to output unit c
w_{jn}	connection weight from input unit n to hidden unit j
ω	dimension of the parameter space
$\mathbf{d}$	search direction vector
η	step size (scalar)
τ	iteration step
E	total error function
E^k	local error function
∇	gradient
∂	partial derivative
δ_j	local error of the j-th hidden node
δ_c	local error of the c-th output node
S	set of training patterns
$\mathcal{F}$	analytically unknown mapping from input to output space
$\in$	element of
$:=$	defined as
$\mathbf{\Phi}$	single hidden layer feedforward network function
$\mathbf{\Phi}_c$	c-th element of $\mathbf{\Phi}$
z_j	activation of the j-th hidden unit
net_j	net input to the j-th hidden unit
net_c	net input to the c-th output unit
$\mathbf{x}$	N-dimensional vector, element of space X^N
$\mathbf{y}$	C-dimensional vector, element of space Y^C
x_n	n-th component of $\mathbf{x}$

y_c	c-th component of $\mathbf{y}$
x_0	bias signal
$\gg$	much greater than
β	scalar parameter
$\mathbf{H}$	Hessian matrix
$\mathbf{I}$	identity matrix
$\mathbf{g}$	gradient vector
$\cong$	approximate

Manfred M. Fischer, Professor and Chair at the Department of Economic and Social Geography at the Wirtschaftsuniversität Vienna as well as Director of the Institute for Urban and Regional Research at the Austrian Academy of Sciences, received the M.Sc. and Ph.D. degrees in geography and mathematics in 1974 and 1975, both from the University of Erlangen (Germany), the habilitation degree in 1982 from the University of Vienna. He is author of more than 150 publications. Dr. Fischer has been a leader in the field of spatial analysis and modelling for some 20 years. Specific research areas of current interest include issues of spatial function approximation and learning by computational neural networks, with applications to remote sensing and interregional telecommunication data.

Petra Staufer received the MA and Ph.D. in geography in 1993 and 1997, both from the University of Vienna and a postgraduate degree in geoinformatics in 1993 from the Technical University of Vienna. Her research interests include neural network modelling and spatial data analysis in the context of geographical information processing (GIS) and remote sensing. She is an assistant professor in the Department of Economic and Social Geography at the Wirtschaftsuniversität Vienna.