Chemometrics and Intelligent Laboratory Systems, 22 (1994) 165–189
Elsevier Science B.V., Amsterdam

Tutorial

Using artificial neural networks for solving chemical problems

Part I. Multi-layer feed-forward networks

J.R.M. Smits, W.J. Melssen, L.M.C. Buydens and G. Kateman

Laboratory for Analytical Chemistry, Faculty of Science, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands)

(Received 25 January 1993; accepted 27 April 1993)

Abstract

Smits, J.R.M., Melssen, W.J., Buydens, L.M.C. and Kateman, G., 1994. Using artificial neural networks for solving chemical problems. Part I. Multi-layer feed-forward networks. *Chemometrics and Intelligent Laboratory Systems*, 22: 165–189.

This tutorial focuses on the practical issues concerning applications of different types of neural networks. The tutorial is divided into two parts. In the first part, an overview of the general appearance of neural networks is given and the multi-layer feed-forward neural network is described. In the second part, the Kohonen self-organising feature map and the Hopfield network are discussed. Since the multi-layer feed-forward neural network is one of the most popular networks, the theory concerning this network can easily be found in other references (B.J. Wythoff, *Chemom. Intell. Lab. Syst.*, 18 (1993) 115–155) and is therefore only described superficially in this paper. Much attention is paid to the practical issues concerning applications of the networks. For each network, a description is given of the types of problems which can be tackled by the specific neural network, followed by a protocol for the development of the system. It is seen that different neural networks are suited for different kinds of problems. Application of the networks is not always straightforward; a lot of constraints and conditions have to be fulfilled when using neural networks properly. They appear to be powerful techniques, but often a lot of experience is needed. In this paper some guidelines are given to avoid the most common difficulties in applying neural networks to chemical problems.

CONTENTS

1. Introduction 166
2. Problem domains 168
   2.1. Memory 168

Correspondence to: W.J. Melssen, Laboratory for Analytical Chemistry, Faculty of Science, Catholic University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen (Netherlands).

SSDI 0169-7439(93)E0035-3

2.2. Generalisation ... 168
2.3. Optimisation ... 168
2.4. Data reduction ... 168
3. Types of neural networks ... 168
3.1. Basic ingredients ... 169
3.2. Overview ... 169
3.2.1. Modeling ... 169
3.2.2. Self-organisation ... 169
3.2.3. Optimisation ... 169
4. Multi-layer feed-forward networks ... 170
4.1. Introduction ... 170
4.2. Theory ... 170
4.2.1. Structure ... 170
4.2.2. Signal propagation ... 171
4.2.3. Representation of problem and solution ... 171
4.2.4. Training the network ... 172
4.2.5. Testing the network ... 173
4.3. Aspects of use ... 173
4.3.1. Types of problems ... 173
4.3.1.1. Binary outputs ... 173
4.3.1.2. Continuous outputs ... 174
4.3.2. Other methods ... 174
4.3.2.1. Neural networks and linear discriminant analysis ... 175
4.3.2.2. Neural networks and principal component analysis ... 176
4.3.2.3. Neural networks and standard modeling techniques ... 178
4.3.3. Protocol ... 178
4.3.3.1. Data acquisition ... 178
4.3.3.2. Data selection ... 179
4.3.3.3. Data preprocessing ... 183
4.3.3.4. Network design ... 183
4.3.3.5. Training and testing ... 184
4.3.3.6. Output interpretation ... 188
Acknowledgement ... 189
References ... 189

1. INTRODUCTION

Chemometrics is the subdiscipline of analytical chemistry which concerns the design and selection of optimal measurement procedures and experiments and the extraction of as much relevant information as possible from chemical data. The fast development of analytical chemical instrumentation together with the need for more quality control leads to more and more data and to the demand for advanced data interpretation methods.

Traditionally, mathematical and statistical methods are used for data processing and interpretation. Standard numerical techniques, however, are incapable of solving some of the more complex problems. For such complex problems nowadays other methods are also used such as expert systems [1,2]. Expert systems combine by means of an inference process the theory underly-

ing a specific problem and available human expertise, e.g. heuristics. Unfortunately human expert knowledge often is very hard to acquire and expert systems are also still limited to restricted domains. Another method which has recently gained interest is the genetic algorithm technique [3–5]. This is a powerful optimisation technique, but it can only handle limited, though complex, problems and a lot of experience is needed to apply it.

Artificial neural networks have been developed initially as models for their biological counterparts. The computerised version of this model is well suited for performing typically human tasks, such as memorising objects, recognising (symbolic) patterns, generalising, estimating parameters and making decisions. These properties seem promising for overcoming some of the shortcomings of the more 'traditional' data interpretation techniques. For these reasons neural networks are being more frequently used by researchers as well as practitioners [6].

Probably one of the earliest descriptions of some of the main ideas of (biological) neural networks is found as far back as 1890, in a psychology book written by James [7], and the foundations of artificial neural networks are perhaps given by McCulloch and Pitts [8] in their paper of 1943. These authors tried to understand the functioning of the nervous system by defining primitive information processing elements, which were based on mathematical logic, that form abstractions of the functional properties of biological neurons and their connections. In 1949 Hebb [9] described a learning rule which was derived from observations done in neurophysiological experiments on biological neural networks. Remarkably, the learning rule embedded in the bulk of the current artificial neural networks is based on this so-called Hebbian learning rule.

At that time research was still theoretical because there was no sophisticated computer technology available. With the increasing availability of computers, the neural network models could be simulated and tested 'in practice'. A famous example is the Perceptron, developed by Rosenblatt [10]. It was the first precisely specified, computationally oriented neural network and it was an impetus for the growth of research on (artificial) neural networks. Increasingly more scientists devoted their time to neural network research and the capabilities of neural networks were believed to be tremendous. This rather exaggerated expectation, together with the scientific anxiety proclaimed in newspapers *, created an atmosphere in which the book of Minsky and Papert [11] could flourish. In this book the authors show the severe shortcomings of Perceptrons. The book, which predicts the uselessness of neural networks, has had a very negative impact on neural network research, which caused loss of research funding.

Fortunately, a few scientists were not discouraged and their persistency resulted in a final breakthrough: the development of a learning rule, i.e., back-propagation, for more complex networks which were capable of dealing with more complex (non-linear) problems than was the Perceptron. This learning rule was developed almost simultaneously in three places. A detailed description of the rule and parallel distributed processing (PDP) is given in a two-volume book by Rumelhart, McClelland and the PDP research group [12]. This revival has led to an expansion of the research on artificial neural networks and, as a consequence, these networks are being more frequently applied to chemical problems. However, the appearance of present artificial neural networks has very little in common with the original biological neural networks.

Research has led to the development of different types of neural networks. They are all composed of units, neurons, and connections between them. These units act in parallel and locally, and together they determine the global behaviour of the network. Most networks are trained or initialised with examples. Once a network has been trained, it may be used if it fulfills the requirements specified in advance. The latter may be verified by presenting a set of test examples to the network and monitoring the network's performance.

* *Frankenstein Monster designed by Navy Robot that Thinks*, headline in an Oklahoma newspaper, 1962.

2. PROBLEM DOMAINS

Neural networks may be used for different kinds of problems. Basically, there are four main types of problems/applications where neural networks could be useful: (auto-)associative memory, generalisation, optimisation and data reduction. In the following discussion, the terms problem and solution are used to refer to the overall, abstract problem and solution. The terms input object and output object are used to refer to a specific instance of the problem and its associated solution, respectively. The terms input pattern/vector and output pattern/vector are used to refer to the numerical representations of the input and output object, respectively.

2.1. Memory

A neural network may be used as a memory, i.e., to recall stored patterns. If a data set with examples (input patterns together with their associated output patterns) is memorised, the network should be capable of recalling the correct output pattern when the corresponding input pattern is presented again. If the input pattern and the associated output pattern are identical, the memory is called auto-associative. A noisy or incomplete pattern is presented to the network to obtain a noise-free or complete pattern.

Usually such an associative network is initialised or trained with a set of examples taken from the data base. After this procedure, the network reflects or models the association between each input–output pattern combination. The complexity of the model is not important. It does not matter whether each specific association is memorised or whether a more abstract relation is built that is valid for more input–output pattern pairs. Neural networks suitable for this task are perceptron-like networks (Section 4 and Part II, Section 3) and Hopfield-like networks (Part II, Sections 2 and 3).

2.2. Generalisation

If the network can not only recall output patterns previously stored during a training or initialisation phase, but can also predict output patterns associated with input patterns it has never seen before, the network is said to generalise. In this case the model that the network has built based on the training examples has to be more general. It should not only memorise the relation between specific input–output pattern pairs, as in an associative memory, but it should model such a relation for an entire domain. This is a much more difficult task and it imposes constraints on the design of the network and the composition of the training set. Neural networks which can perform these tasks are perceptron-like (Section 4 and Part II, Section 3) and Kohonen-like networks (Part II, Sections 1 and 3).

2.3. Optimisation

Another class of neural networks is capable of optimising non-optimal situations, given some constraints and a measure, i.e., a cost or energy function, to express the quality of the solutions. Neural networks suitable for this task are Hopfield-like networks (Part II, Sections 2 and 3).

2.4. Data reduction

A pattern (representing some object) consists of a number of variables. This number may be high and for various reasons it may be desirable to reduce it, e.g., variables may be relatively unimportant or highly correlated. Neural networks suitable for this task are perceptron-like (Section 4 and Part II, Section 3) and Kohonen-like networks (Part II, Sections 1 and 3).

3. TYPES OF NEURAL NETWORKS

Since the foundation of artificial neural networks, a variety of different types has been developed. The choice of the network type depends on the particular problem to be solved. Before discussing different types of networks, some basic building blocks will be listed.

3.1. Basic ingredients

In general, artificial neural networks are composed of the following basic building blocks:

- Units (neurons or processing elements). Units may be associated with some objects in different ways. Each object may be associated with exactly one unit or with a set of units together. The last possibility is termed parallel distribution. The way the units are organised, e.g., in layers or other configurations, is important.
- Pattern of network connections. The units are connected with each other by network connections. Units and their connections together determine the structure of the network. Via these connections the units are able to send/receive signals to/from each other or the outside world.
- Weights of connections. Every connection is associated with a connectivity strength, a weight. These weights play an important role in the propagation of signals through the network. Every signal passing a connection is multiplied by the weight associated with this connection. The weights contain information, in a distributed sense, on the relation between the ensemble of input and output patterns.
- Activity of units. The activity of a unit depends on the signals the unit receives and influences the final signal the unit will send.
- Activity function. In each unit the incoming signals are processed to form a net input. This net input, together with the actual activity, determines the new activity of the unit via the activity function.
- Transfer function (output function). The output signal of a unit is determined by applying the transfer function to the activity of that unit. The pattern of outputs of all units which emerges determines which object is involved.
- Learning rule. Optimal weight values must be found for the network to function properly. These values are achieved by training the network, i.e., by adapting the weights according to some learning rule.
- Environment. The environment of a neural network is made up of the problem and the solution. Both problem and solution impose constraints on the structure of the network.

Not every neural network contains all of these building blocks.

3.2. Overview

In Section 2 several problem domains were mentioned in which neural networks could be applicable. There are many different networks, each with its own capabilities and limitations. In this overview a description is given of the basic types of neural networks. Globally, neural networks may be subdivided into three basic types, suited for modeling, self-organisation, and optimisation, respectively.

3.2.1. Modeling

If a neural network is used for modeling, it has to build a model of the relation between the given problem and solution, i.e., it has to be able to transform an input pattern to the associated output pattern (pattern association). These types of networks are trained in a supervised way. They are provided with input–output pattern pairs and extract the model from these examples. Once the network has built this model, it may be used, after some validation procedure, to predict output patterns for new input patterns. An example of this type of network is the multi-layer feed-forward neural network (Section 4).

3.2.2. Self-organisation

If only examples of input patterns without their output patterns are available, so-called self-organising neural networks may be applied which are able to organise themselves by the presented training data. Such neural networks will reflect the structure present in the training set. The network is not forced to give a specific solution, since this solution is not known. The networks are trained in an unsupervised way. An example of this type of neural network is the Kohonen self-organising feature map (Part II, Section 1).

3.2.3. Optimisation

A third type of neural network may be used to solve optimisation problems. Such a network may be initialised with a far from optimal situation and the network will optimise the given situation

by itself, provided that the network has some measure for the quality of the solutions. An example of such a network is the Hopfield network (Part II, Section 2).

4. MULTI-LAYER FEED-FORWARD NETWORKS

4.1. Introduction

The multi-layer feed-forward (MLF) neural networks, also called multi-layer perceptron or back-propagation neural networks, are presently popular and are used more than other types, for a wide variety of problems. If a solution for a problem cannot be derived directly, e.g., mathematically or numerically, from a description of a problem, an indirect path has to be found to model the relation between the problem and its solution. The application of a neural network is based on the assumption that such a relation indeed does exist.

A relation between problem and solution may be quite general, e.g., the simulation of a production process (where the problem is defined by the process parameters and the solution by a description of the product) or the prediction of chemical or physical properties of a chemical compound. A MLF neural network is a powerful system, often capable of modeling such (complex) relations. This enables the use of a MLF network for predicting an output object for a given input object. The network builds a model based on examples with known outputs, a process which is referred to as supervised learning. No information about the relation to be modeled is given explicitly to the network. It must extract this relation solely from the presented examples, which together are assumed to contain implicitly the necessary information for this relation.

4.2. Theory

4.2.1. Structure

A multi-layer feed-forward neural network (Fig. 1) consists of three or more layers of units: one input layer, one output layer and one or more intermediate (hidden) layers. All the units in one layer are connected with all the units in the next layer (feed-forward). The number of input and output units depends on the representations of the input and the output objects, respectively. Notwithstanding these and other restrictions, much variety in the network structure is possible.

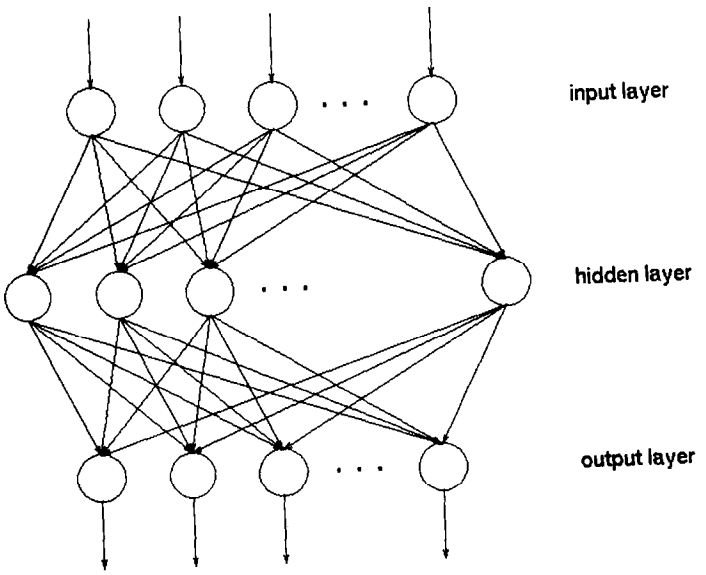

Fig. 1. A MLF network with one input layer, one hidden layer and one output layer.

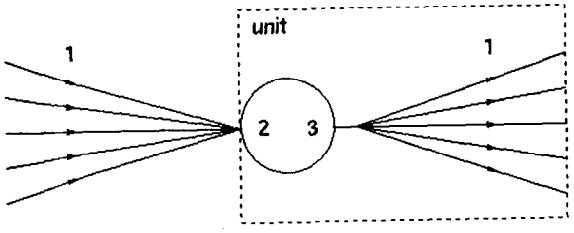

Fig. 2. Basic processing element of a neural network. See Section 4.2.2 for an explanation of the different phases.

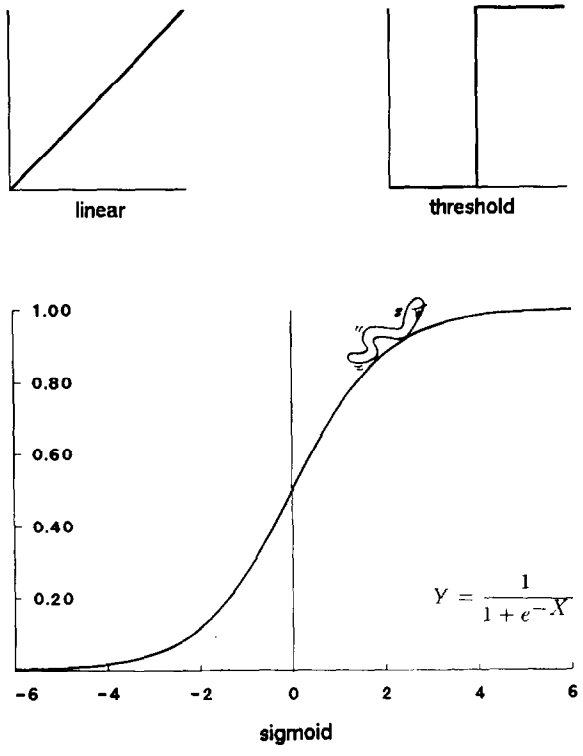

Fig. 3. Different transfer functions. "The actual 'intelligence' exhibited by the most sophisticated artificial neural network is below the level of a tapeworm" [13].

4.2.2. Signal propagation

Units and their connections form the backbone of a neural network. The general appearance of a unit with its connections is shown in Fig. 2. In this figure, three phases can be distinguished:

1. A unit receives and sends signals from and to other units or the outside world via the connections. Every signal is weighted by a weight factor that is associated with the connection.

2. The received weighted signals together determine the net input to the unit. For the units in a layer the net input of unit j is given by

$$net_j = \sum_i w_{ji} o_i \qquad (1)$$

in which the index i refers to the units in the previous layer, w_{ji} is the weight from unit i to unit j, and o_i indicates the output of unit i. This net input then determines the activity of a unit via the activity function. However, in most networks the activity of unit j is given by

$$act_j = net_j \qquad (2)$$

and in the following net_j is used to denote the activity of unit j.

3. The activity of the unit determines the transmitted signal (output) of the unit via a transfer function. Many transfer functions may be used, e.g., a linear function, a threshold function or a sigmoid function (Fig. 3). A sigmoid function that is used often is given by

$$o_j = f(net_j) = \frac{1}{1 + \exp[-(net_j + \theta_j)]} \qquad (3)$$

in which θ_j is a bias term which influences the horizontal offset of the function. The bias, θ_j, may be treated as the weight from an extra input unit to unit j. This extra input unit has a fixed output value of 1, and θ_j may be trained as a regular weight.

The weights play an important role in the propagation of the signals through the network. They establish a link between the input pattern and its associated output pattern and are said to contain the knowledge of the neural network about the problem–solution relation.

4.2.3. Representation of problem and solution

Since the network has to extract the relation between problem and solution from examples, the representation is a very important issue (Fig. 4). As much information as possible has to be retained both upon translation of the input object to an input pattern for the network and upon translation of the output pattern to the output object. Both the input object and its associated

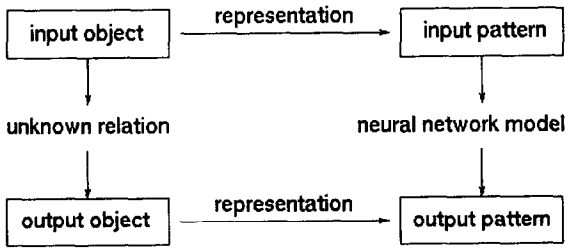

Fig. 4. Relation between problem and solution.

output object are represented by an array of variables (a pattern or vector). An example thus consists of an input and an output pattern, a pattern pair. Every variable is associated with a unit. Since each unit is defined to have a specific meaning, the number of variables and their meaning have to be the same for each pattern pair.

Different applications of neural networks often require different representations. When a neural network is used for classification, each output unit may be defined to be associated with a specific class. In this case the representation of the output objects (classes) may be a binary one. If the network is used for calibration, however, a continuous output representation is usually required. The input pattern is presented to the input units of the neural network (one variable value per input unit). The signal is propagated through the network to the output units, which give the output pattern.

4.2.4. Training the network

As was stressed before, the functioning of a neural network is highly dependent on the way the signals are propagated through the network. This signal propagation, in turn, is determined by the weights of the unit-to-unit connections. In general, the weight setting is not known beforehand and, therefore, initially the weights are given a random value. The process of updating the weights to a correct set of values is called training or learning.

A correct weight set usually is achieved by means of supervised learning. During training, examples consisting of input–output pattern pairs are forced iteratively upon the initially untrained network. Each time an input pattern is presented, the output pattern given by the network is compared to the known, desired, output pattern, and the difference is used to adjust the weights in small steps. The presentation of patterns from the training set continues until the network gives the correct answer for each input pattern of the training set, possibly within some predefined allowed error, or after a predefined number of presentations of all the examples.

This training procedure is called the back-propagation learning rule [12]. The difference between the desired output pattern and the actual output pattern (the error) is a function of the weights (Fig. 5). The back-propagation learning rule tries to locate the minimum error in this weight space, by including a gradient descent approach. This error is given by

$$E = \tfrac{1}{2}\sum_j (d_j - o_j)^2 \qquad (4)$$

in which the fraction $\frac{1}{2}$ is included for mathematical reasons, and d_j and o_j are the desired output and the actual output of unit j, respectively.

The adaptation of the weights is done in a backward fashion. According to the back-propagation learning rule, first the weights to the output layer are adapted, next the weights between units in two consecutive intermediate layers are adjusted, and finally, the weights of the input layer to the second layer in the network are modified. The adaptations are given by

$$\Delta w_{ji} = \eta \delta_j o_i \qquad (5)$$

in which Δw_{ji} denotes the adaptation of the weight from unit i to unit j in the next layer, o_i is the output of unit i and η is the learning rate. The error correction term, δ_j, depends on the

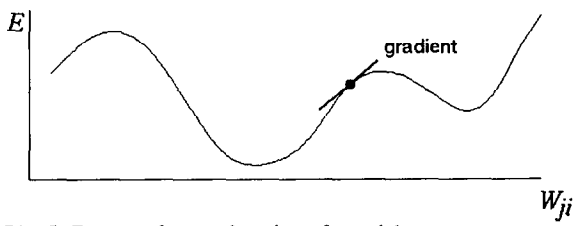

Fig. 5. Error surface as function of a weight.

layer index. If the layer is the output layer, δ_j is given by

$$\delta_j = (d_j - o_j) f_j'(net_j) \quad (6)$$

in which d_j and o_j represent the desired output and the actual output of unit j, respectively, and $f_j'(net_j)$ is the derivative of the transfer function, $f_j(net_j)$ (Eqn. 3), with respect to its argument. If the layer is a hidden layer, δ_j is given by

$$\delta_j = f_j'(net_j) \sum_k \delta_k w_{kj} \quad (7)$$

in which k refers to the units in the next layer.

The learning rate is an important network parameter because it strongly determines the progress of the training procedure. If it is chosen too small, the convergence of the weight set to an optimum is accurate, but very slow, and the network might get stuck in a local optimum. If the learning rate is high, on the other hand, the system might oscillate. To damp possible oscillations, often a momentum term, α, is invoked. In that case Δw_{ji} is

$$\Delta w_{ji}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ji}(n) \quad (8)$$

The training set must have enough examples to be representative for the overall problem. The training phase can be time consuming depending on, amongst other things, the network structure, the number of examples in the training set, and the number of iterations. However, it only has to be done once for a particular problem.

4.2.5. Testing the network

After training, the performance of the network must be tested. This is done with a test set consisting of examples other than the training set, taken from the original data set. In the testing phase the input patterns are fed to the network and the desired output patterns are compared with those given by the neural network. The (dis)agreement of the two output pattern sets gives an indication of the performance of the trained network. Often the performance of such systems is expressed in two percentages: Recognition and Prediction. Recognition and Prediction are the number of correct output patterns divided by the total number of pattern pairs present in the set. In the case of Recognition the set of pattern pairs comes from the training set. In the case of Prediction the pattern pairs do not come from the training set. When the performance meets the requirements specified in advance, the network is ready for real analysis purposes.

4.3. Aspects of use

Developing a neural network system is said to be an art. Many parameters have to be set and this is often done based to some extent on heuristics. In the following paragraphs, some of these heuristics are discussed. This Section is not meant to give an overview of all possible situations one might encounter during the development of a neural network system, nor to solve all problems that may occur. It it is merely meant to be a guideline to avoid many of the typical traps.

4.3.1. Types of problems

Although the characteristics of the MLF neural networks impose certain restrictions on their use, they still have enough diversity to be useful for solving many different kinds of problems. Depending on the chosen representation of the output objects, different types of network applications may be defined. The output of the network may consist of binary or continuous values, each suited for different fields of applications. Qualification requires binary outputs and quantification requires continuous outputs. If somewhat more complex networks are used, combinations of these types of outputs may be used. However, these are not discussed in this paper.

4.3.1.1. Binary outputs. The input pattern of the network consists of variables which together characterise the input object. The output pattern of the network consists of binary values. To obtain the binary values, some sort of threshold transfer function is necessary for the output units, e.g., a sigmoid function.

If the network is used for feature detection, it is requested to indicate the presence or absence of specific features of an object. The output pattern indicates the presence/absence of the features, e.g., in the case of three features an output

of (1 0 0) indicates the presence of the first feature and absence of the second and third, (1 1 0) indicates the presence of the first and second feature and absence of the third, and so on. More output values may be equal to 1 simultaneously, thus indicating the presence of a combination of features. An example of the use of MLF networks for qualification is given in ref. 14 where the interpretation of infrared spectra with neural networks is described. The network is requested to indicate the presence/absence of different functional groups in a molecule, based on infrared spectra.

If binary outputs are used, the network may also be used for classification tasks. For such tasks, the network is requested to distinguish between different classes of objects. The output pattern of the network gives the class of the object. To indicate a class, different representations may be used. Often each output unit is associated with one class and binary values are used to indicate to which class the object belongs, i.e., in the case of four classes an output of (1 0 0 0) indicates the first class, (0 1 0 0) the second, etc. Since an object is designated to belong to precisely one class, only one output unit is supposed to have a unitary value.

If it is desirable that the number of units is less than the number of classes, gray coding [5] or binary coding may be used, e.g., (0 0 1), (0 1 0), (0 1 1) etc., indicating class one, two, three, etc., respectively. Even one *continuous* output unit may be used to indicate the classes, e.g., the value 1, 2, or 3 indicating class one, two, or three, respectively. However, using the last mentioned output representation, it is implicitly assumed that there exists some sort of ordering of the object classes. Moreover, the network might mix up different classes more easily and the interpretation of the network output would be less straightforward. Usually, one output unit per class is the best choice. An example of the use of MLF networks for classification is given in ref. 15. In this paper algae, characterised by flow cytometer data, are classified.

Classification tasks occur quite often. If each class is associated with only one unit, the interpretation of the network output is simplified. In the following the term classification refers to this specific combination of task and representation. The term qualification will be used to refer to all other tasks performed by networks with binary outputs.

4.3.1.2. Continuous outputs. Again, the input pattern of the network consists of a number of variables which characterise the object. The output pattern now consists of one or more continuous values. This means a threshold is not required (no distinction between binary values) so the transfer function of the output units may be a linear one. The transfer function of the units in the hidden layer is still a sigmoid to enable the network to model non-linear relations.

If continuous outputs are used, not only the presence/absence of specific features may be indicated, but also quantitative information on these features. The interpretation of the output is straightforward: the output values of the output units provide the (scaled) quantitative information. An example of such an application is given in ref. 16. If the solution domain is the same as the problem domain, the network may be used for data reduction or as an associative memory. For these networks, during training the input and the output patterns are identical. If the network is used for data reduction, the number of hidden units is smaller than the number of input and output units in order to achieve a reduction of the number of variables. Since the input may be transformed or encoded to values for the hidden units, and since these values in turn may again be transformed or decoded to the output values, the outputs of the hidden units may be used as a reduced representation for the input pattern. The network may also be used as an associative memory, which gives the correct pattern if a distorted (e.g., noisy) or incomplete pattern is presented.

In the following, the term auto-association is reserved for the case where the input and output of the network are identical. The term quantification refers to all other tasks performed by networks with continuous output values.

4.3.2. Other methods

As described above, MLF neural networks are used to perform classification, calibration, and

data reduction tasks that can also be tackled by a broad scale of chemometrical techniques including discriminant analysis, multivariate regression, and principal component analysis. Research on artificial neural networks includes the investigation of the relationship between these networks and other techniques. This will make it possible to position neural networks properly within the field of other more established statistical and numerical techniques. Since neural networks are based on different principles, this relationship is not easy to find.

The following discusses some similarities and differences between MLF neural networks and some widely accepted chemometrical techniques. This is not an exhaustive overview and is intended to provide the reader with a feeling of the difference in problem-solving strategies followed by these techniques.

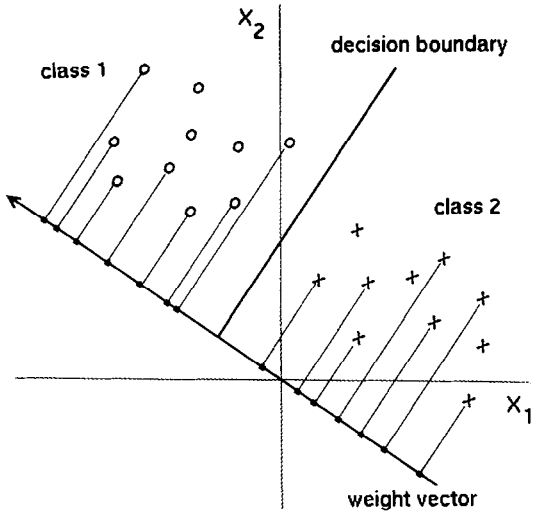

Fig. 6. Division of objects in two classes by linear discriminant analysis.

4.3.2.1. Neural networks and linear discriminant analysis. A well-established technique to perform supervised classification is statistical linear discriminant analysis (LDA) [17]. From a set of examples, each characterised by a number of variables, LDA tries to divide the object space, spanned by the variables, into two or more distinct classes. Fig. 6 depicts a two-class problem in which the objects are characterised by two variables, say x_1 and x_2. The goal of LDA is to find the weight vector as drawn in Fig. 6. The projections of the objects on the weight vector are also shown. LDA tries to find the weight vector for which the within-class variance of the projections is minimal and the between-class variance of the projections is maximal. The decision boundary, i.e., the discriminant line, is defined to be perpendicular to this weight vector. Unknown objects can then be classified in one of the two classes according to this decision boundary.

Fig. 7a shows a neural network that is used for classification. It can be envisaged that the neural

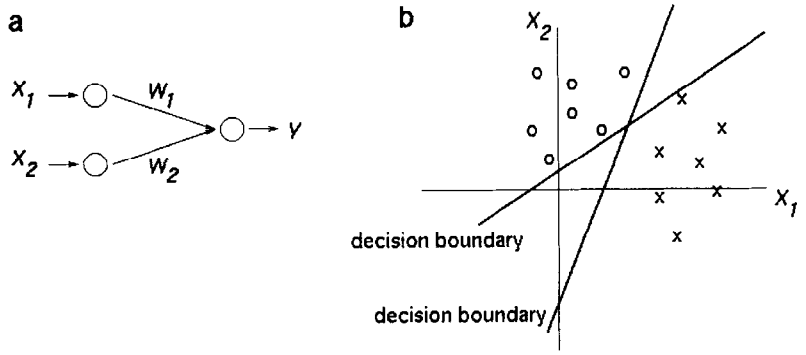

Fig. 7. (a) A two-layer neural network. (b) Division of objects in two classes by the neural network. (a) The two input units are flow-through units, the transfer function of the output unit is a threshold function. The net input for the output unit is given by $net = w_1x_1 + w_2x_2$. The output of the output unit, y, is given by $y = f(net) = 1$ if $net \geq t$, and 0 if $net < t$, in which t denotes the threshold. The value of y indicates to which class (represented by '0' and '1') the object belongs. The equation for the decision boundary is given by $w_1x_1 + w_2x_2 = t$. Two possible decision boundaries that might be revealed from the neural network are depicted in (b).

network also defines a weight vector and a derived decision boundary from a set of known objects. The difference between LDA and the network lies in the criterion that is used to select the weight vector. The neural network tries to find that weight vector for which the error (Eqn. 4) is a minimum. Initially the weight vector is randomly chosen. During the training phase the weight vector is modified, according to the back-propagation learning rule, in order to minimise the network output error. Depending on the initial position of the weight vector, different decision lines may be revealed. In Fig. 7b some decision boundaries are drawn that are equally probable regarding the criterion of the neural network.

The difference in strategy between the neural network and LDA might result in different performances in some situations. LDA is a parametric technique that is based on the assumption that all classes possess equal variances. When this is not the case, LDA does not perform optimally. The MLF network is, on the other hand, a non-parametric method and does not require such an assumption. Its performance is not influenced by unbalanced variances in the classes. When the assumptions of LDA are met, then the solution produced by LDA is the optimal one. However, the actual solution of the network and, hence, its performance, depends on the initial random settings of the weights. This is a disadvantage of the neural network in comparison with LDA.

Another non-ideal situation is the presence of outliers in one or more classes. This is illustrated in Fig. 8. LDA is not able to cope with these outliers while the network is flexible enough to find a weight vector as shown in the figure. The reason behind this difference in performance lies in the fact that the network, at the end of the training phase, focuses solely on boundary objects to find the discriminant line while LDA focuses on the 'mean object' of each class. Hence, objects deviating too much from the 'mean object' deteriorate the performance of the LDA technique. Neural networks are not hampered as much in this situation. This explains why MLF neural networks usually perform better with 'difficult' data sets. If, however, all assumptions of LDA are met, this technique is to be preferred because it yields a unique, optimal solution. Table 1 summarises the comparison of both techniques.

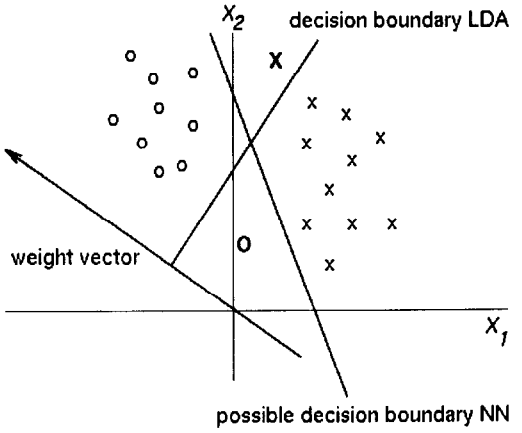

Fig. 8. Presence of outliers in the data set. Shown are decision lines obtained with linear discriminant analysis and the two-layer neural network of Fig. 7, respectively. Bold symbols nearby the ordinate indicate outliers of both classes.

4.3.2.2. Neural networks and principal component analysis. Data reduction is another major application field for neural networks. It is therefore interesting to see how this is related to principal component analysis (PCA), the best known data reduction technique in chemometrics. In Fig. 9 an auto-associative network is depicted schematically. The desired output pattern equals the input pattern. In the hidden layer a number of hidden units is chosen that is smaller than the number of input and output units. The network has to squeeze its input through the bottleneck formed by the hidden layer to the output layer in such a way that each input pattern is reproduced as best as possible. If an appropriate number of hidden

TABLE 1

Comparison of LDA and MLF

LDA	MLF
Parametric	Non-parametric
Assumptions:	No assumptions
Distribution	
Variances	
Focuses on 'mean object'	Focuses on boundary objects

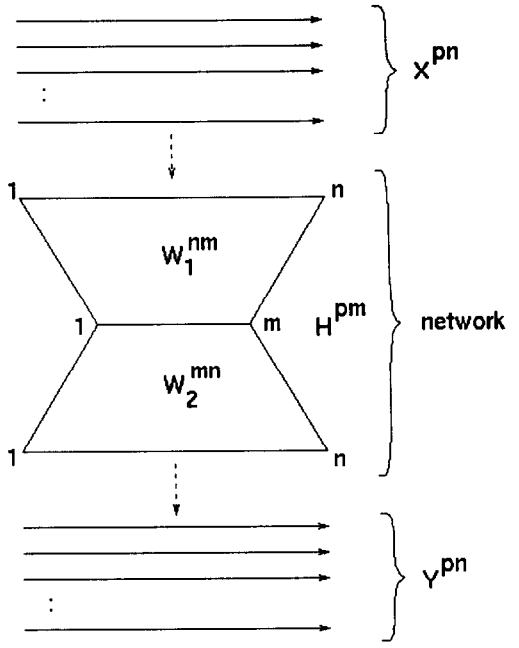

Fig. 9. An auto-associative network. The data set consists of p input patterns with each n variables. Since it is an auto-associative problem, the desired output patterns equal the input patterns: $\mathbf{Y}^{pn} = \mathbf{X}^{pn}$. The neural network has n input units, n output units and m hidden units, with $m < n$. For the set of input patterns the resulting set of hidden unit outputs is given by $\mathbf{H}^{pm} = \mathbf{X}^{pn}\mathbf{W}_1^{nm}$, whereas the set of outputs of units in the output layer are expressed as $\mathbf{Y}^{pn} = \mathbf{H}^{pm}\mathbf{W}_2^{mn}$.

units is chosen, the hidden layer forms an optimal reduced representation of the input, allowing a nearly perfect retransformation of the input pattern. The criterion of the back-propagation learning rule to set the weights of the network is to minimise the output error given in Eqn. 4.

When we compare this with PCA, the equivalence is immediately clear. PCA decomposes the **X** matrix, in which the rows comprise the input patterns, into a matrix, **S**, of scores and a matrix, **L**, of loadings. This is done so that an m-dimensional reproduction of the **X** matrix, denoted by $\mathbf{X}(m)$, is the best possible one, according to the criterion that $[\mathbf{X}(m) - \mathbf{X}]^2$ is minimised. This may be summarised by

$$\mathbf{X} = \mathbf{S}^{pn}\mathbf{L}^{nn}$$

and

$$\mathbf{X}(m) = \mathbf{S}^{pm}\mathbf{L}^{mn}$$

where p and n denote the number of objects and variables, respectively. Compared with the equation of the neural network output

$$\mathbf{Y}^{pn} = \mathbf{H}^{pm}\mathbf{W}_2^{mn}$$

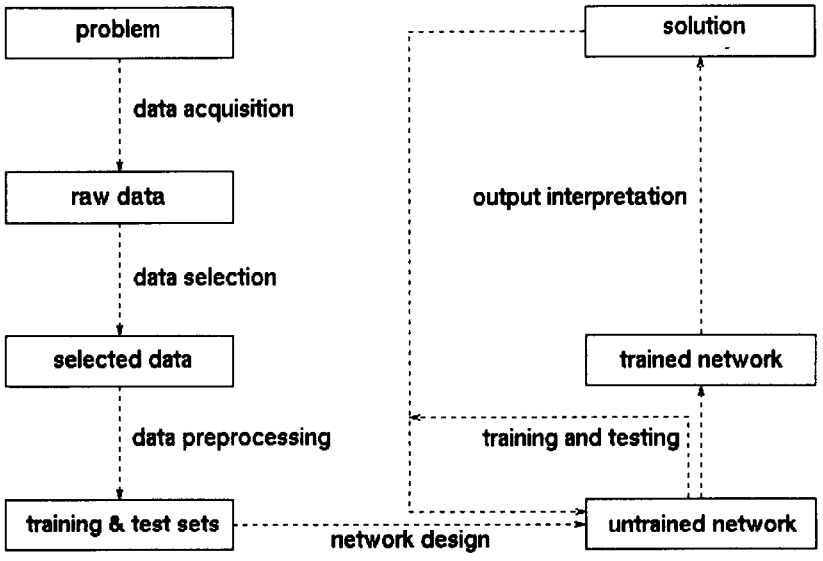

Fig. 10. General protocol for developing MLF networks.

this demonstrates that a neural network with m hidden units should produce an m-dimensional PCA solution, except for a rotation matrix T. This can be expressed as

$$\mathbf{Y}^{pn} = \mathbf{X}(m)$$

and

$$\mathbf{H}^{pm}\mathbf{W}_2^{mn} = \mathbf{S}^{pm}\mathbf{T}\mathbf{T}^{-1}\mathbf{L}^{nm}$$

where the property of rotation matrices, i.e., $\mathbf{T}\mathbf{T}^{-1} = 1$, is used. In ref. 18 it is demonstrated that a network that is initialised with the PCA solution cannot improve this solution. Special network architectures have been developed to produce exactly the PCA solution [19–21]. These networks are beyond the scope of this Tutorial.

4.3.2.3. Neural networks and standard modeling techniques. Artificial neural networks have been proven to be able to model complex non-linear input–output relations, where other techniques fail. Different studies have compared the performance of neural networks with partial least squares (PLS) and principal component regression (PCR) [22]. When a highly non-linear relation underlies the examined problem, usually neural networks outperform these classical techniques. This fact initiated or stimulated research on other non-linear techniques, e.g., non-linear PLS. The comparison and the relationship between them is still under investigation and it is expected that in the near future the first results will be published.

4.3.3. Protocol

MLF neural networks may be used for many kinds of problems for which several approaches exist. Nevertheless, a general protocol may be given, consisting of a sequence of actions and stages. This protocol is depicted schematically in Fig. 10. The actions are described in more detail in the following paragraphs. Since some of the situations described below occur in different parts of the protocol, some redundancy is inevitable.

4.3.3.1. Data acquisition. Since the networks are trained under supervision, examples have to be available. Before gathering any data, a good representation for the input and output objects has to be found. The input and the output patterns of a neural network consist of variables (one per unit) and thus both the problem and the solution have to be translated (see paragraph 4.2.3). The representations depend on the given problem and the desired solution (Fig. 11).

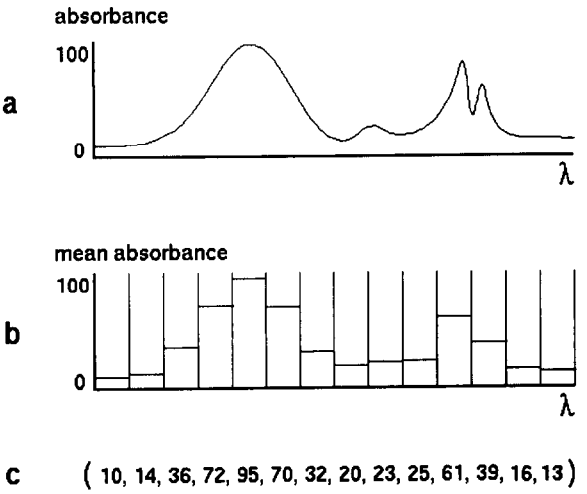

Fig. 11. Example of a representation of a spectrum. (a) The input object (a spectrum). (b) Division in intervals. Intervals too small: too many variables (noise); intervals too broad: loss of information. (c) The resulting (unscaled) input pattern.

Often the decision concerning the representations is based on knowledge about the problem and experience with neural networks. Finding good representations may be a tedious task. A choice must be made about the number of variables that is used to represent the input and the output object. Usually a compromise has to be made: not too few variables (units) to retain as much information as possible and not too many to prevent the loss of the network's capability to generalise. The latter situation might occur especially if there are only a few examples available.

Once the representations are decided, data may be collected. Since the network has to extract the relation between problem and solution from examples, the data set must be as representative as possible. A good strategy for obtaining the data is desirable, but often this phase is not controlled by the designer of the network. Usually some data are already available and collect-

ing more or other data often is too expensive or time-consuming. Despite the requirements mentioned earlier, in most cases one must make do with what one has.

4.3.3.2. Data selection. If an abundant amount of raw data is available, a selection has to be made. The data must be representative and there should be enough data available to prevent the network from overtraining. Also there must be enough data to allow subdivision of the data set into different sets for training and testing.

Distribution of the data. First of all, the examples with which the network will be trained have to be 'representative'. This does not mean 'representative for the real world', but 'representative for the problem'. The data have to encompass as much as possible the complete domain on which the model has to be built, so that the network interpolates instead of extrapolates. Not only the distribution of the data points is of importance, but also their ratio of appearance. For example, if the network has to distinguish between two classes of objects, in the 'real world' these two classes

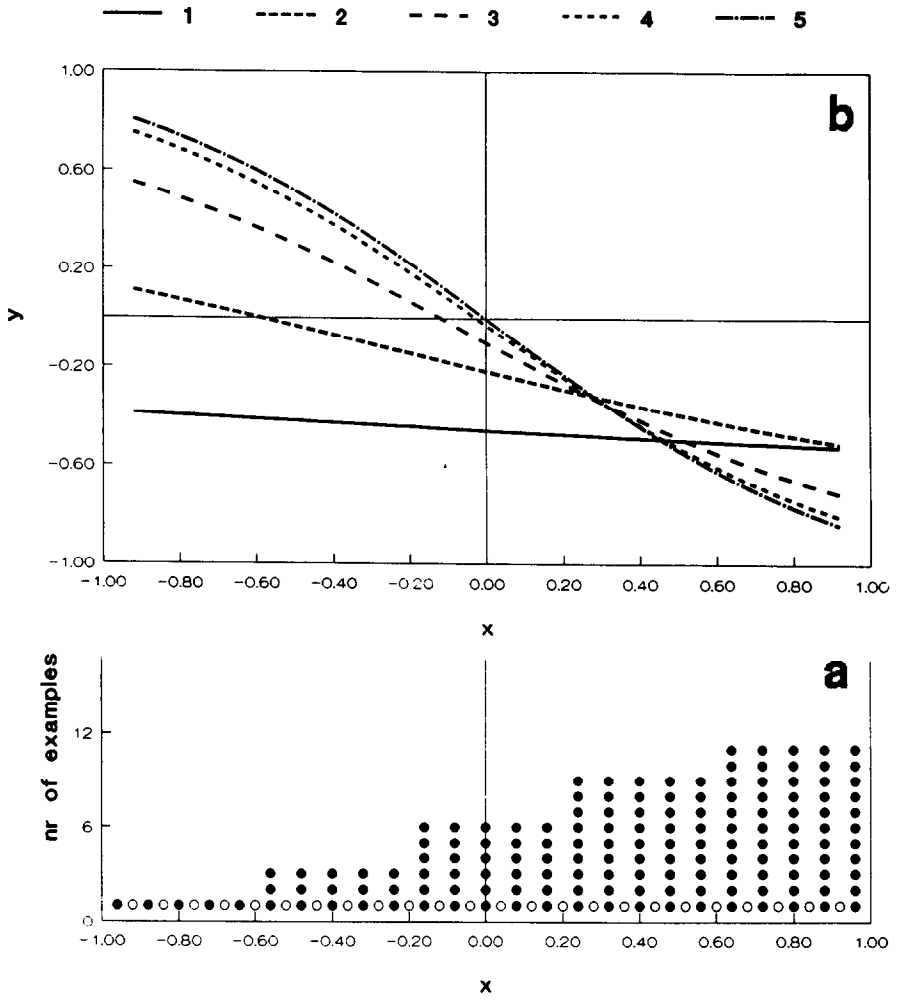

Fig. 12. Learning the function $y = -x$ with an unbalanced training set. (a) Composition of (•) training set and (○) test set. (b) Line given by network at different stages (1 to 5) of the training phase.

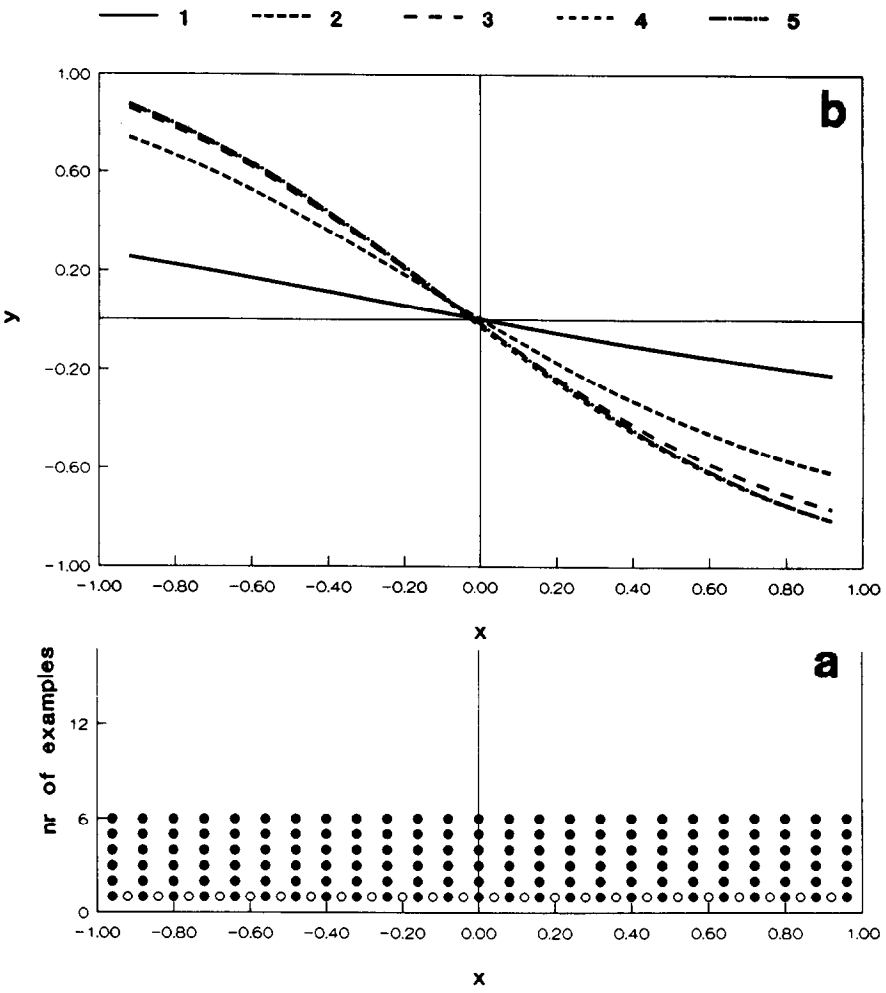

Fig. 13. Learning the function $y = -x$ with a balanced training set. (a) Composition of (•) training set and (○) test set. (b) Line given by network at different stages (1 to 5) of the training phase.

may appear in different amounts, say 1 : 50. If the network is trained with a data set which contains examples of both classes in this same ratio, i.e., representative for the real world, the network will have problems learning to recognise the class with only a few examples. If both classes have to be learned equally well, balancing of the training set is a prerequisite. The same problem might arise if the network is being used for other types of problems, like quantitative analysis.

A simple example will be given to illustrate the effect of using a data set that is not representative for the problem. If the function $y = -x$ has to be trained, a simple network may be used with one input, two hidden * and one output unit. The output unit is equipped with a linear transfer function. The training and test set both consist of pairs of (x, y) values. In Fig. 12a an unbalanced training set and the test set are given. If the network is trained with this unbalanced training set, it has more difficulties learning the left part of the line than the right part of the line. In Fig.

* To solve this linear problem, a neural network without a hidden layer, or even another method, could have been used as well. However, for illustration purposes a hidden layer was added.

12b five test set results are presented, obtained after one, two, three, four and five iterations of the training set, respectively. It is seen that the network indeed is biased towards the right part of the line. If the training set is balanced, like the one presented in Fig. 13a, the network exhibits less problems (Fig. 13b). Of course, for this simple problem the line will also be learned with the unbalanced training set after only a few iterations more. However, in real-world situations it is not always possible to recover from performance problems arising from unbalanced training sets by just taking more iterations. The part of the model where many examples were present in the training set might already be overfitted while other parts are still not trained well enough.

Overtraining of the network. If a network is overfitted (overtrained), it acts like a memory. In such cases, the network will not learn the general features inherently present in the training set during training, but it will learn more and more of the specific details of the particular examples. Thus the network gradually looses its capability to generalise. This can happen only when the training set exhibits specific features which are not to be included in the model. Such a situation often occurs if only a few noisy examples are available, especially if there are many units. The risk of overtraining then is high. To give an example, a model has been built with the training set presented in both Fig. 14a and b. The same network structure given in the previous example has been used. In Fig. 14c the curve that is learned by the network is shown at different stages in the training process. It is seen that, during training, the model changes from an almost straight line (s1) to a relatively strongly curved line (s4). If the test set presented in Fig. 14a is used, this network is said to be overtrained. It tries to reproduce the specific training set as

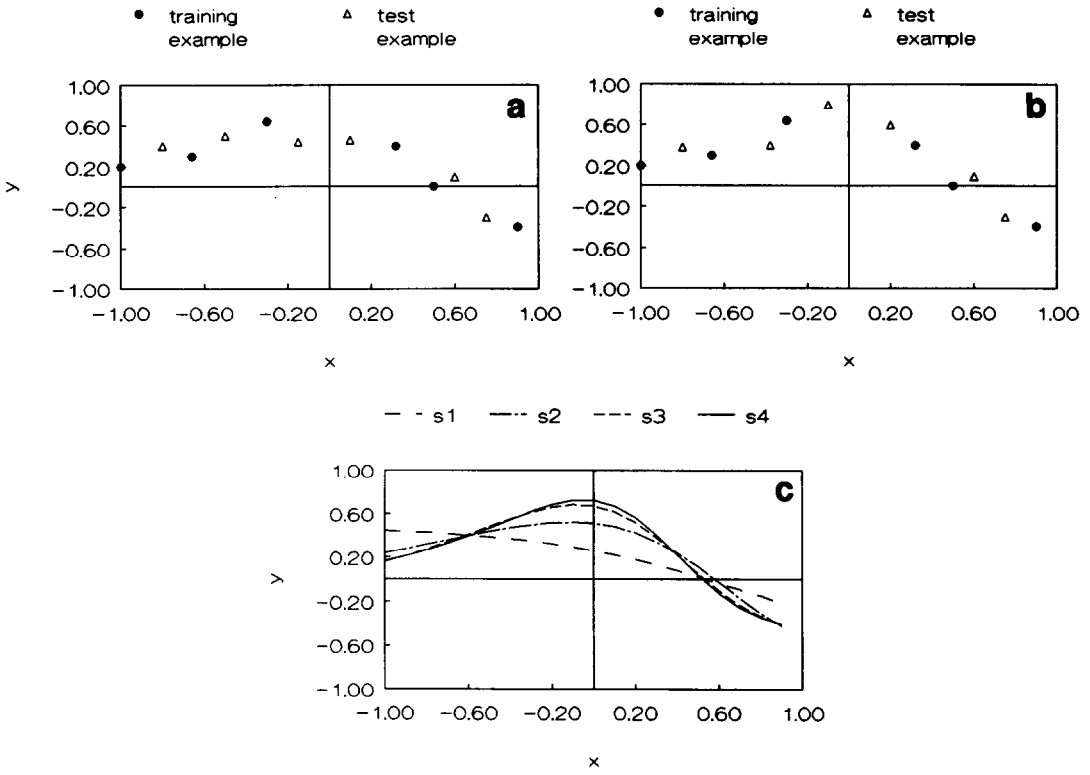

Fig. 14. (a) and (b) Two different data sets with the same training examples. (c) Model built by a network at different stages (s1 to s4) during the training.

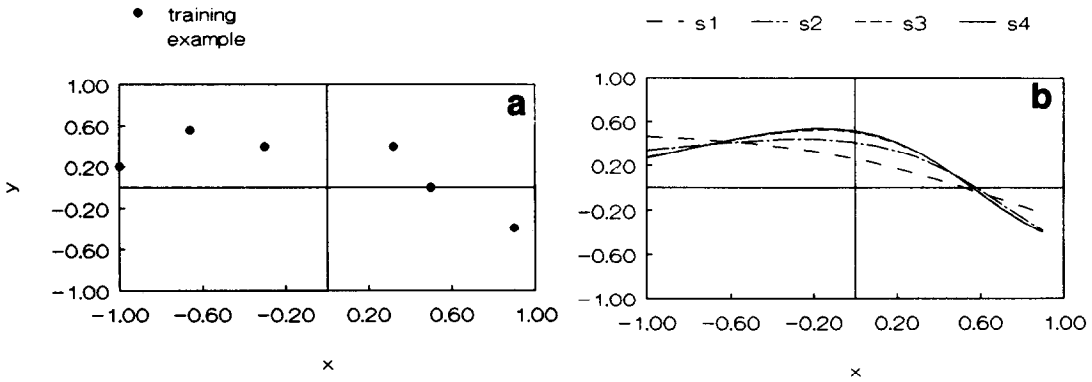

Fig. 15. (a) Training set. (b) Model built by a network at different stages (s1 to s4) during the training.

best as possible by learning the noise too. However, if another test set, presented in Fig. 14b, is used, the *same* network model is not overtrained, but for this combination of training and test set it is a good model. Overtraining happens when the network is able to build a more complex model based on the training set alone than the model the training and test set together appear to define. For the training set presented in Fig. 15a a network with the same structure as used in the previous example is not able to build a very complex model (Fig. 15b). Although this training set contains an equal amount of noise as the previous one, it leads less easy to overtraining. The situation of overtraining is comparable to fitting a curve with a polynomial of too high an order.

Subdivision of the data. Another decision that must be made is the subdivision of the data set into different sub-sets which are used for training and testing. If enough examples are available, the data set may be split by some fraction (e.g. 50/50% or 67/33%) into the training and test sets. The training set still has to be large enough to be representative of the problem and the test set has to be large enough to allow validation of the network.

If there are not enough examples available to permit splitting of the data set into a representative training and test set, other strategies (like cross-validation) may be used (see Fig. 16). In this case the data set is split in N different ways into a training set and a (usually smaller) test set, respectively. The same network structure may now be trained and tested N times with the N different pairs of training and test sets. Every pattern is thus used once as a test pattern in one of the N procedures. The results of these tests together enable determination of the performance of the network. If the test set consists of only one example, this strategy is called the leave-one-out method. Of course, more test data is

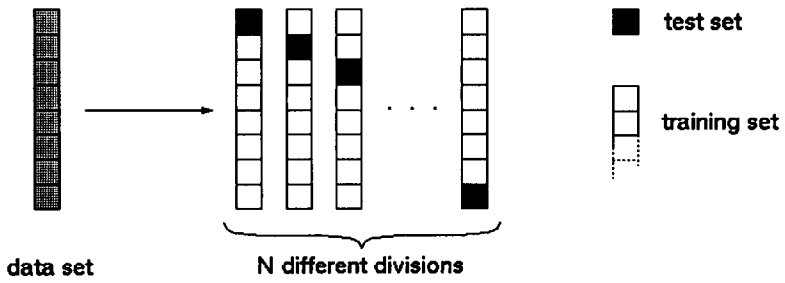

Fig. 16. The cross-validation technique.

preferable, but the cross-validation method makes it possible to use smaller amounts of data.

4.3.3.3. Data preprocessing. After selection of the training and test sets, some data preprocessing might still be desirable. Often the data are scaled before use, especially if different variables have different ranges of values. If some of the variables have relatively high values, these variables might dominate the model. If the input values are too large (especially in combination with large weights), the net input (Eqn. 1) for the units may end up in the tails of the sigmoid function, where the derivative is very small. Since according to the back-propagation learning rule the weights are adapted in proportion to this derivative, this may lead to a static situation, also called paralysis of the network. Scaling of the input brings the net input within the dynamical range of the sigmoid transfer function. One may scale the input, the output or both. Different preprocessing strategies exist for binary and continuous values.

Binary values. If binary values are used, scaling is not necessary. Of course, one must always ensure that the desired output values are in the output domain of the transfer function used. If 0 and 1 are the limits of the sigmoid transfer function used, exhaustive training will be necessary to obtain output values close to 0 or 1. If 0.1 and 0.9 are given as correct values instead of 0 and 1, the network can put more emphasis on training 'difficult' examples where the output is still far from correct instead of bringing the almost correct ones to perfection.

Continuous values. For scaling of the data different methods may be used. The choice depends on the type of problem and the chosen representation of the input and output objects. Methods which are often used with good results are autoscaling and range-scaling. The data may be scaled per variable, but if the variables are highly correlated (as in a spectrum) scaling the values per object (spectrum) should be used.

4.3.3.4. Network design. If the training and test sets are generated, a choice has to be made about the network structure and its parameters, e.g., the number of input, hidden and output units; the transfer function; the weight initialisation; the learning rate, η; the momentum, α; and the number of iterations. Usually first a rough estimation is made for the different parameters to define some point from where the optimisation process starts. Several network parameters certainly are not independent of each other, but to some extent a univariate optimisation procedure is sufficient to obtain an initial design.

Number of units. The number of input and output units is of course equal to the number of variables with which the input and output objects, respectively, are represented. Many heuristic guidelines for the choice of the number of hidden units exist [23], ranging from 2 times the number of input units via 2/3 of the number of input or output units (whichever is less) to 1/2 times the number of input plus output units. However, since the optimal number of hidden units depends so strongly on the nature of the problem and on the chosen representations for the input and output objects, the authors feel it is not safe to rely exclusively on heuristics. It is better to scan a range of possibilities. This will lead quickly to a good approximation of the number of hidden units.

If the nature of the problem is linear, hidden units are not necessary and a network with no hidden layer at all will also do the job *. If the problem is non-linear, some minimal number of hidden units is required. If less hidden units are taken, the performance of the network drops sharply. If more hidden units are taken, the performance increases slightly to a limiting value or even decreases again. Usually, the number of hidden units is best chosen to be not too much above the minimum amount necessary, even if the performance does not deteriorate if more hidden units are used. If more hidden units are used, training does not only take more computing time, but also the performance might fluctuate more. If PCA is performed on the data set, the number of significant principal components often

* For a linear problem, one of the standard (chemometrical) techniques might be favoured.

gives an indication of the minimum number of hidden units necessary.

There is still an ongoing debate as to whether a three- or four-layer neural network, i.e., with one or two hidden layers, respectively, is able to model *any* arbitrary non-linear tranformation between the input and output objects (see, e.g., refs. 23 and 24). In our experience, a variety of chemical problems (e.g., refs. 15 and 16) can be solved by a three-layer neural network. In some circumstances application of a four-layer network resulted in a much faster convergence and therefore might save a substantial amount of computing time. However, in none of these cases did the four-layer network outperform the three-layer one. When a three-layer network appeared to be incapable of modeling a particular problem–solution relation, a four-layer network failed to solve this problem as well.

Transfer function. The input units are flow-through units meaning that the transfer function can be expressed as $f(x)=x$. For the hidden units usually a sigmoid transfer function is taken (Fig. 3). If a linear transfer function is used, the network will only be able to model linear relations and a network with no hidden layer at all will also suffice. A hidden layer with units possessing linear transfer functions therefore only makes sense if the network is used to perform data reduction. For the output units the choice of the transfer function depends on the type of problem for which the network is used, as described in Section 4.3.1.

The weights. Initially, the weights are usually assigned random values within a certain range around zero (e.g., -0.3 to 0.3) in order to bring the net input of each unit in the network into the dynamical range of the sigmoid function. This is to prevent the network from becoming paralysed.

The learning rate and momentum. After the design of the network, values for the learning rate and the momentum still have to be chosen. If the transfer function of the output layer is a sigmoid, the output of a unit is limited by definition, no matter how extreme the input value. In this case a rather high value for η may be chosen: between 0.7 and 0.9. If, however, the transfer function is linear, then there is no limit on the output of a unit and high values for η often cause the network to oscillate or, even worse, diverge. In this case η is typically chosen at least a factor of ten smaller, i.e., below 0.1. The operation of the network is also much more influenced by η when a linear transfer function is applied.

Previous changes in the weights are taken into account with the momentum term, which smoothens to some degree the learning behaviour and thus limits the danger of oscillations or divergence. The momentum term is usually set between 0.3 and 0.6. Its influence is not as predominant as that of the learning rate.

Number of iterations. A presentation of the entire training set, followed by a presentation of a test set, is defined as one iteration. The choice of the number of iterations that has to be performed is based on trial experiments, in which the performance of a network as function of the number of iterations is monitored.

4.3.3.5. Training and testing. The initial network design described in the previous section is to some extent based on experience. A rough optimisation is obtained by carrying out the small loop given in Fig. 10. When the initial network structure and parameter settings are chosen, a rough optimisation may be accomplished by monitoring the performance of networks having slightly different structures and parameters. Obviously, the best strategy is to perform an experimental design to obtain the behaviour of the network as a function of its structure and parameter settings. This requires training and testing of a substantial number of networks, a procedure which might be very time consuming.

The functioning of a network depends on the chosen network structure and parameters. The normalised standard error (*NSE*) may serve as a measure of the performance of a network and facilitates the comparison of performances of different networks. Based on this *NSE*, some typical phenomena will be discussed.

The normalised standard error. The normalised standard error is defined by

$$NSE = \frac{1}{PJ} \sum_{p} \sum_{j} (d_{pj} - o_{pj})^2 \qquad (9)$$

in which P denotes the number of output patterns in the data set and J is the number of output units. The indices p and j refer to the pth output pattern, and jth output unit, respectively, whereas d_{pj} and o_{pj} represent the desired and obtained output value, respectively, of unit j on pattern p.

This *NSE* is not always the best indication of the performance of a network. An error which is very high for one pattern (e.g., an outlier), but almost zero for the other patterns, leads to the same *NSE* as a moderate error for all patterns. The first situation is preferable, but the *NSE* does not reflect this difference. Also, a relatively high *NSE* does not necessarily mean a bad performance. For example, if a unit is desired to give 0 or 1, but this unit consistently gives output values around 0.3 for the 0 values and values around 0.7 for the 1 values, the *NSE* may be large. In that case, however, the performance of the network is perfect when during the interpretation of the output (see below) a threshold of 0.5 is chosen. In general, the trend of the *NSE* is of more importance than each of its individual values.

For a first optimisation, however, the *NSE* is sufficient. If a rough idea for a good network structure and parameter settings is obtained, another criterion than the *NSE* may be used for further optimisation. In this case the next two steps in Fig. 10, data analysis and interpretation, are included in this process (indicated by the large loop in the figure) to determine the performance and perhaps further optimise the network. This process is described later in Section 4.3.3.6.

Some typical examples. In the remainder of this paragraph, some typical examples will be discussed where the *NSE* is used as a first indication of the network performance. The remedies that are presented for the problems considered in

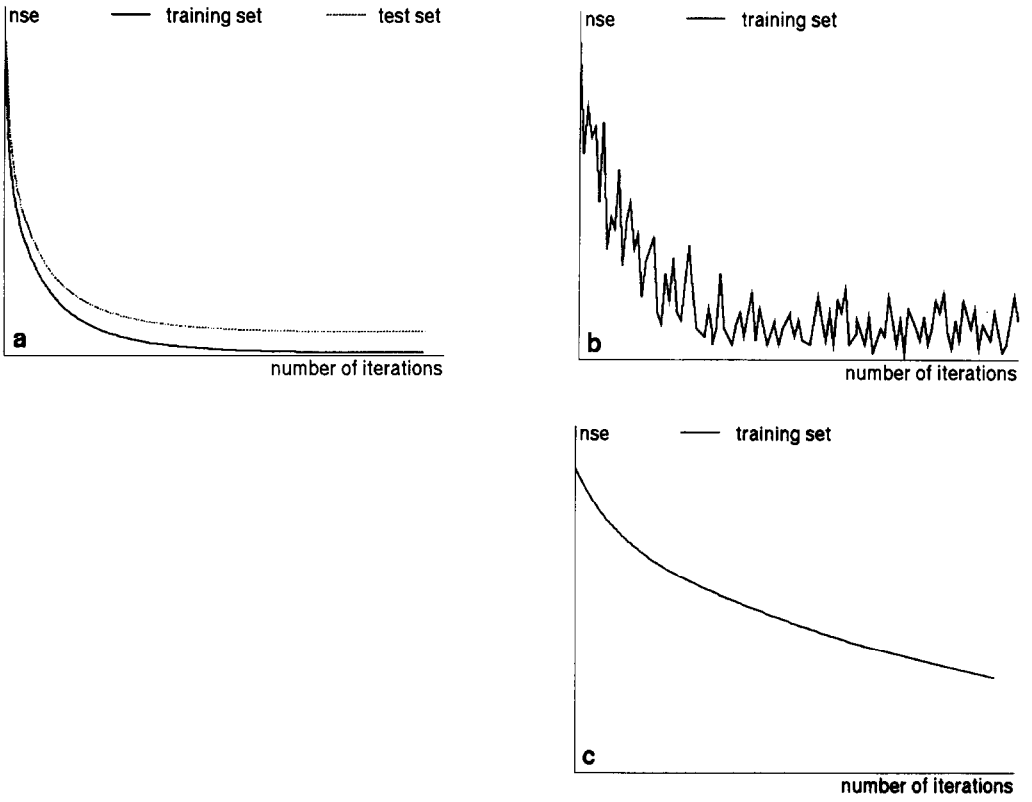

Fig. 17. Training behaviour of a network, indicated by the progress of the *NSE*; effect of the learning rate on the training. See text for more details.

the following examples are mostly based on changes in the network structure and the parameter setting, since in this phase the data set is considered as fixed. Many problems may be avoided or solved by taking more and/or other data, but since this is often not possible one must make the best of it.

A successful training session is shown in Fig. 17a. Both the *NSE* for the training set and the test set converge to a minimum value. If the learning rate, η, is chosen too large, a training session as shown in Fig. 17b may occur (the test set is omitted). The *NSE* more or less converges but fluctuates a lot as a function of the number of iterations. Here, the network wanders around a minimum, and the functioning of the network depends highly on the precise moment when training ends. In the worst case, the network might even diverge instead of converge. If, on the other hand, η is too small, too many iterations are necessary to achieve a convergence of the network (Fig. 17c). If enough time is available, this does not have to be a problem, but a small η increases the possibility of getting the network trapped in a local minimum. A good strategy is to decrease η as a function of the number of iterations: use a large η initially to enable the network to locate the neighbourhood of the minimum and a decreasing η to let the network settle down in the exact position of the minimum. In the neighbourhood of the minimum, usually the derivative is very small. Since the adaptation of the weights is proportional to this derivative, it would be advantageous to increase the value of η again if the network is close to the minimum, but unfortunately, often this reversal point is difficult to determine.

A training session like that in Fig. 18a might indicate subsequently that the network is stuck in a local optimum, is moving across a relatively flat part of the hypersurface described by the *NSE* in the weight space, or is paralysed. The latter situation may be avoided by taking smaller random weights and/or invoking another scaling procedure for the inputs. To prevent getting stuck in local optima, one may consider other learning algorithms than back-propagation. However, most learning algorithms have this risk of encountering local optima in common. Another remedy is given by increasing η, or alternatively, starting the same training session multiple times, with different initial random weights, hoping that in one of the runs a deeper minimum of the *NSE* is reached. Of course, one never knows whether the deepest minimum that is found is a local or a global one, unless the *NSE* of both the training and test set becomes negligibly small. However, if the network meets the requirements, it is ready for use.

Even if the network converges to a local minimum, the question still remains as to whether the network would have been able to escape from it if more iterations were used. For instance, Fig. 18a might be just the left part of Fig. 18b, in which the network realises an escape from a local mini-

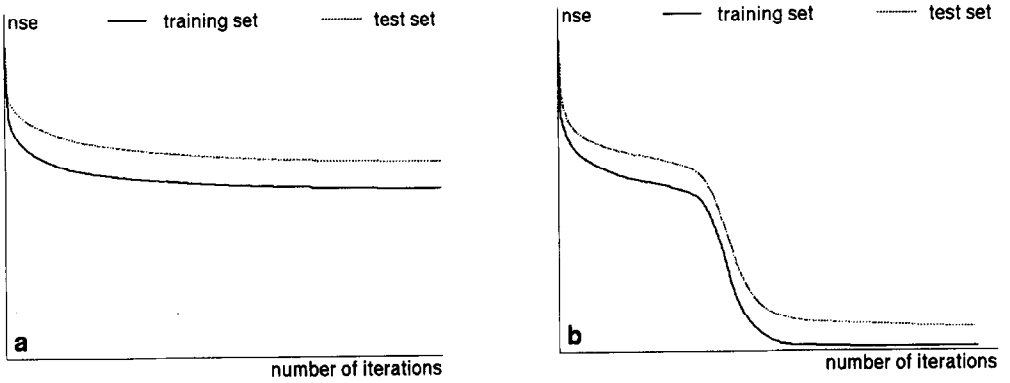

Fig. 18. Training behaviour of a network, indicated by the progress of the *NSE*; effect of paralyzation of the network or shapes of error surface on the training. See text for more details.

mum or reaches the edge of an elongated hypersurface in weight space.

Too many iterations or hidden units may cause overtraining of a network (Fig. 19a). Such a network acts more and more like a memory, capable of recalling the presented training examples (a decreasing *NSE* for the training set), but losing its capability to generalise (an increasing *NSE* for the test set after some minimum). Overtraining especially occurs when the training set contains too few examples. As a consequence, the noise present in the patterns of the training set is learned by the network. In this case the training has to be ended at the minimum of the *NSE* for the test set. If this minimum is not satisfactory, a remedy might be to use less hidden units. Of course, ending the training at the minimum is not an elegant solution. Using more or other data often is a better approach.

Closely related to overtraining is the situation shown in Fig. 19b. In this case a gap appears between the *NSE* curves for the training and test set after just a few iterations. No minimum for the *NSE* for the test set can be observed and limitation of the number of iterations will not do any good. The gap between the two *NSE* curves indicates that apparently the training set represents a different input–output relation than the test set. This happens, for instance, when the test set contains more outliers or noisy patterns than the training set. If the original data is split in another way into a training and test set this phenomenon might disappear.

The gap between the two curves also appears if there are not enough data available to allow a meaningful subdivision into a training and a test set. Each individual set does not contain sufficient information to describe the input–output relation and in that circumstance merging of the data of both the training and test set might be necessary. A test set containing only a few patterns may be taken (e.g., according to the leave-one-out method) to see if more training examples helps to solve the problem.

The gap may also appear when an oversized network together with an input and/or output representation of a too high dimension is chosen. For example, suppose that an arbitrary number of input patterns, each consisting of 200 variables, is presented to a three-layer network which possesses twenty hidden units. After determination of the *NSE* curves it appears that a gap is manifest. Then a resizing of the network, e.g., taking ten hidden units, together with a lower dimensional input representation, e.g., 100 variables per input pattern, might solve this problem.

It should be noticed that when the scales of the ordinates in the previously mentioned figures are changed, one figure might look like another indicating that it is not always straightforward to

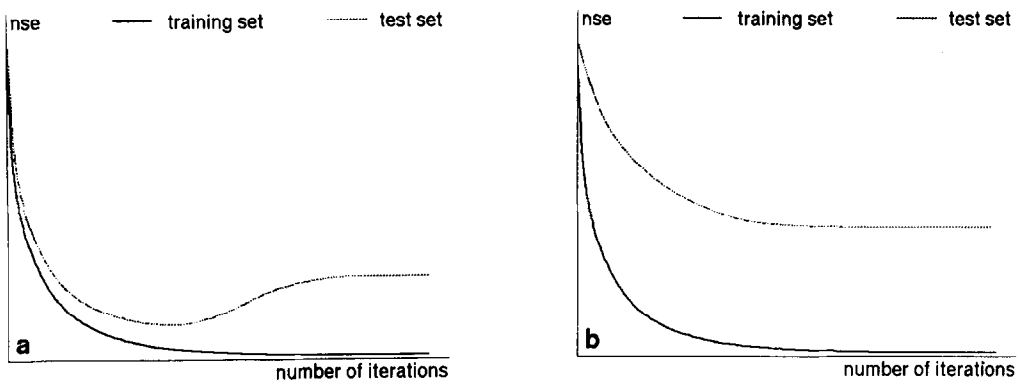

Fig. 19. Training behaviour of a network, indicated by the progress of the *NSE*; training set and test set appear to describe different relations. See text for more details.

discriminate between the aforementioned situations.

4.3.3.6. Output interpretation. In this stage of the protocol an optimal *NSE* that suffices is assumed. The training and test set are presented again to the trained network, but now without adapting the weights any further. The output produced by the network for each example of the training and test set is now available. If the output patterns have been scaled, these may be scaled back to their original values, in order to facilitate the interpretation of these outputs. Different criteria may be used to interpret the output of the network, depending on the type of the problem and the chosen output representation.

Classification. A large variety of classification criteria exists which depend among other things on the chosen output representation. If each output unit is associated with a single class, the following three classification criteria might be appropriate:

1. The input pattern is said to belong to the class corresponding to the output unit with the highest value. This implies that each input pattern is assigned to a class and unknowns are not possible.

2. The first criterion is extended with the condition that the highest value must exceed a predefined threshold, which may be different for each of the output units. If none of the output values exceeds this threshold, the input pattern is said to be unknown.

3. The second criterion may be extended further with the requirement that each of the other output values has to be a predefined amount lower than the highest one. Unknowns may then be defined as patterns for which none of the output units exceeds the threshold. Doubtful cases may be defined as patterns for which a unit does exceed the threshold, but other outputs are not a predefined amount lower than the highest one.

Qualification. If the network is used for qualification and binary outputs indicate the absence (0) or presence (1) of certain features, the output interpretation must be performed *per unit*. Obviously, a 'one-highest' criterion cannot be used in this case. The outputs of a single unit for all patterns in the training or test set may be distributed as in Fig. 20a. In this case only one threshold is necessary. The particular feature is defined to be absent if the output is lower than the threshold and present if the output exceeds this threshold. In this way, doubtful cases and unknowns are not defined.

If the outputs are distributed according to Fig. 20b, two thresholds may be used. If the output unit is below the lower threshold, the associated feature is said to be absent. If the output unit is above the higher threshold, the associated feature is defined to be present. Values between these thresholds indicate a doubt with the specific feature. Here, unknowns are not defined. In this kind of analysis, three different data sets may be used: a set to train the network, a first test set to determine the best positions of the two thresholds after examination of the output distribution obtained with this set, and a second test set (sometimes referred to as a generalisation set) to validate the performance of the network including the output interpretation step. The position of the thresholds may differ for different units.

In both these cases, the output gives an indication of the confidence one may have in the result from the network. If the value of an output unit is

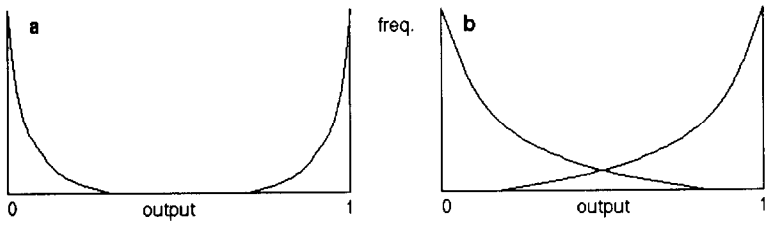

Fig. 20. Two possible distributions of the outputs of a unit for all patterns in a set.

close to 1, one may trust this answer more than the case where the value exceeds the threshold only slightly. In fact, a more fuzzy criterion may be used. Outputs ranging from 0 to 1 may be interpreted as going from 'feature certainly not present' via 'doubt' to 'feature certainly present'.

Quantification. In the case of a neural network with continuous output values, no interpretation criterion is necessary. The value of an output unit itself ought to give quantitative information on the associated feature. The answer provides no clue concerning its certainty.

ACKNOWLEDGEMENT

The authors would like to thank Marco Derksen for his support and helpful discussions.

REFERENCES

1 L. Buydens and P. Schoenmakers (Editors), *Intelligent Software for Chemical Analysis*, Elsevier, Amsterdam, 1993, in preparation.
2 B.A. Hohne and T.H. Pierce, *Expert System Applications in Chemistry* (ACS Symposium Series, Vol. 306), American Chemical Society, Washington, DC, 1989.
3 D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
4 C.B. Lucasius and G. Kateman, Understanding and using genetic algorithms. Part 1. Concepts, properties and context, *Chemometrics and Intelligent Laboratory Systems*, 19 (1993) 1–33.
5 C.B. Lucasius and G. Kateman, Understanding and using genetic algorithms. Part 2. Representation, configuration and hybridization, *Chemometrics and Intelligent Laboratory Systems*, accepted.
6 J. Zupan and J. Gasteiger, Neural networks: A new method for solving chemical problems or just a passing phase?, *Analytica Chimica Acta*, 248 (1991) 1–30.
7 W. James, *Psychology (Briefer Course)*, Holt, New York, 1890, chap. XVI, pp. 253–279.
8 W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5 (1943) 115–133.
9 D.O. Hebb, *The Organization of Behaviour*, Wiley, New York, 1949.
10 F. Rosenblatt, The perceptron: a probabilistic model for information storage and organisation in the brain, *Psychological Review*, 65 (1958) 386–408.
11 M.L. Minsky and S.A. Papert, *Perceptrons, An Introduction to Computational Geometry (Expanded Edition)*, MIT Press, Cambridge, MA, 1988.
12 D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vols. 1 and 2, MIT Press, London, 1986.
13 P.D. Wasserman, *Neural Computing. Theory and Practice*, Van Nostrand-Reinhold, New York, 1989.
14 J.R.M. Smits, P. Schoenmakers, A. Stehmann, F. Sijstermans and G. Kateman, Interpretation of infrared spectra with modular neural-network systems, *Chemometrics and Intelligent Laboratory Systems*, 18 (1993) 27–39.
15 J.R.M. Smits, H.W. Balfoort, L.W. Breedveld, J. Snoek, J.W. Hofstraat, M.W.J. Derksen and G. Kateman, Pattern classification with artificial neural networks: classification of algae, based upon flow cytometer data, *Analytica Chimica Acta*, 258 (1992) 11–25.
16 J.R. Long, V.G. Gregoriou and P.J. Gemperline, Spectroscopic calibration and quantitation using artificial neural networks, *Analytical Chemistry*, 62 (1990) 1791–1797.
17 D.L. Massart, B.G.M. Vandeginste, S.N. Deming, Y. Michotte and L. Kaufman, *Chemometrics: a Textbook*, Elsevier, Amsterdam, 1988.
18 P. Baldi and K. Hornik, Neural networks and principal component analysis: learning from examples without local minima, *Neural Networks*, 2 (1989) 53–58.
19 J. Rubner and P. Tavan, A self-organising network for principal component analysis, *Europhysics Letters*, 10 (1989) 693–698.
20 H. Boulard and Y. Kamp, Auto-association by multi-layer perceptrons and singular value decomposition, *Biological Cybernetics*, 59 (1988) 291–294.
21 K.I. Diamantaras, *Principal Component Learning Networks and Applications*, PhD Thesis, Princeton University, 1992.
22 S. Wold, Nonlinear partial least squares modelling. II. Spline inner relation, *Chemometrics and Intelligent Laboratory Systems*, 14 (1992) 71–84.
23 R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading, MA, 1990.
24 R.P. Lippmann, An introduction to computing with neural nets, *IEEE ASSP Magazine*, 4(2) (1987) 22.