Search-based Algorithms for Multilayer Perceptrons

by

Mirosław Kordos

www.phys.uni.torun.pl/~kordos

A dissertation submitted for the degree
of Doctor of Philosophy

Supervisor: prof. Włodzisław Duch

The Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science
Institute of Computer Science

Gliwice 2005

Thesis

Algorithms based on systematic search techniques can be successfully applied for multilayer perceptron (MLP) training and for logical rule extraction from data using MLP networks. The proposed solutions are easier to implement and frequently outperform gradient-based optimization algorithms.

Abstract

Search-based techniques, popular in artificial intelligence and almost completely neglected in neural networks can be the basis for MLP network training algorithms. There are plenty of well-known search algorithms, however since they are not suitable for MLP training, new algorithms dedicated to this task must be developed. Search algorithms applied to MLP networks change network parameters (weights and biases) and check the influence of the changes on the error function. MLP networks considered in this thesis are used for data classification and logical rule-based understanding of the data. The proposed solutions in many cases outperform gradient-based backpropagation algorithms. The thesis is organized in three parts.

The first part of the thesis concentrates on better understanding of MLP properties. That includes PCA-based projections of error surfaces and learning trajectories, trends and statistics of weight changes and visualization of hidden and output neuron activities. Since the network training is in fact realized by searching for a minimum on the error surface, the knowledge obtained from the error surface analysis can be incorporated in learning algorithms, thus making network training more efficient. Learning trajectories are placed on the error surface. Observing them can also suggest some improvements to the existing learning algorithms or can help with designing new ones. Visualization of the hidden and output neuron activities can suggest possible ways of clustering or removing some training data. Analysis of trends and statistics of weight changes provides more information that can be used to tune the training parameters. Several conclusions drawn from this research are used for designing and optimizing MLP learning algorithms in the second part of the thesis.

The second part of the thesis introduces two search-based MLP learning algorithms: numerical gradient and variable step search algorithm. In contrast to the training algorithms that use analytical gradients, they impose no restrictions on transfer functions, error functions or neural connection structures. In particular computationally cheap, non-differentiable transfer functions can be used. Spurious local minima are a typical problem of algorithms that back-propagate the error to hidden layers. Because the influence of hidden layer weights on the network error is directly checked in search-based algorithms, the direction towards the minimum can be determined in each learning step more precisely. The advantages of search-based methods include fast and reliable convergence, low variance of results obtained with different starting points, low memory requirements and simple implementation of the algorithms because complicated derivatives of the error function are not required. Although local optimization methods, including search-based ones, do not guarantee finding a global minimum for every problem, for the prevailing number of real-world problems the proposed

methods are sufficient. Only in rare cases the use of global optimization methods that require much higher computational effort may be required, giving a greater chance to find optimal solutions for complex problems.

The third part of the thesis presents a search-based approach to logical rule extraction from data using MLP networks with quantized parameters. The network training is quite fast, frequently one training cycle is sufficient and the final network function is converted to logical rules using a simple analysis of the network weights. If needed, the network structure is dynamically adjusted to the dataset properties. Feature selection and data discretization are also automatically performed by the network. Various modifications of the method are presented, each generating a specific form of rules. Depending on the desired information one of the methods can be chosen.

Acknowledgements

First, I want to thank my supervisor prof. Włodzisław Duch for his time, guidance, successful cooperation and many interesting ideas, then prof. Tadeusz Czachórski for his help with organizing my PhD studies, dr Krzysztof Grąbczewski for explanations of some detailed topics, Marek Pyś for his help with Delphi programming issues, Marcus Gallagher whose PhD thesis on MLP error surfaces was a significant inspiration for my research, and all the authors of numerous computer programs and publications that proved useful with my PhD thesis. Last but not least I am grateful to my wife Magdalena and my sons Witold and Szczepan for good conditions for my work.

Original Contribution

The original contribution of this thesis comprises: the detailed analysis of factors influencing MLP error surface supported by PCA-based error surface visualization, the analysis of directions in MLP weight space and two MLP training algorithms based on the above analysis: numerical gradient (NG) and variable step search algorithm (VSS). Additionally, two search-based training methods for special structure MLP networks (SMLP) used for logical rule extraction from data were developed: direct search method and a method based on the modified VSS algorithm.

Contents

Part 1

Properties of Multilayer Perceptrons

1.1. Introduction

An artificial neural network is a general mathematical computing paradigm that models the operations of biological neural systems [Hen 2002]. Research on artificial neural networks was originated in 1943 by McCulloch and Pitts [McCulloch 1943] who proposed the first mathematical model of a neuron. In 1958 Rosenblatt [Rosenblatt 1958] introduced the first neural network known as perceptron. All neural network models that have been proposed over the years, share a common building block known as a neuron and a networked interconnection structure. The most widely used neuron model is based on McCulloch and Pitts' neuron and the most widely used neural network called multilayer perceptron is based on several sequentially connected layers of perceptrons.

In general, neural networks can be divided into feed-forward and recurrent networks. In recurrent networks, the output signals of neurons are by feedback also given as their input signals. In feed-forward networks, an output signal of a neuron has no more influence on its input – the signals are propagated only forward. Multilayer perceptron considered in this thesis belongs to the feed-forward networks.

1.1.1. Neuron Model

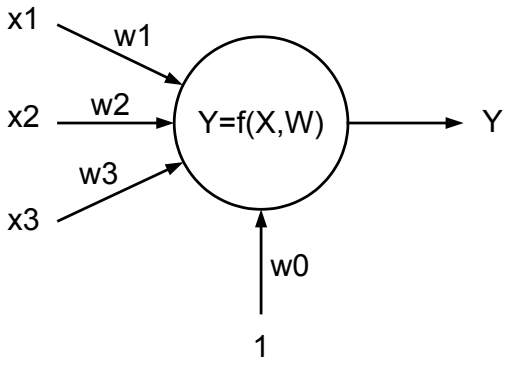

Fig. 1.1. Neuron model.

A neuron consists of two parts: the net function and the activation function. The activation function is also known as transfer function. The net function determines how the input signals are combined inside the neuron.

The most commonly used net function and the only one considered in this thesis is given by the following formula:

$$u = \sum_{i=0}^{N} x_i w_i \tag{1.1}$$

The parameters w are called weights. The weight w_0 is called bias or threshold and its corresponding input signal x_0 always equals 1 and does not form a connection between two neurons as other weights do. In the first and second part of the thesis the term "weight" is used as well for any weight connecting two neurons as for bias.

Table 1.1. Commonly used neural transfer functions.

transfer function	formula	comments
hyperbolic tangent	Y=(1-exp(-βu))/(1+exp(-βu))	
logistic sigmoid	Y=1/(1+exp(-βu))	
threshold	Y=a for u≤0, Y=b for u>0	usually a=-1 or 0, b=1
linear saturated	Y=a for u≤u1, Y=βu for u1<u<u2, Y=b for u≥u2	usually a=-1 or 0, b=1
linear	Y=βu	used only in the output network layer for function approximation tasks, not used for data classification
staircase		not suitable for analytical gradient-based learning algorithms, usually b=1, a=-1 or 0

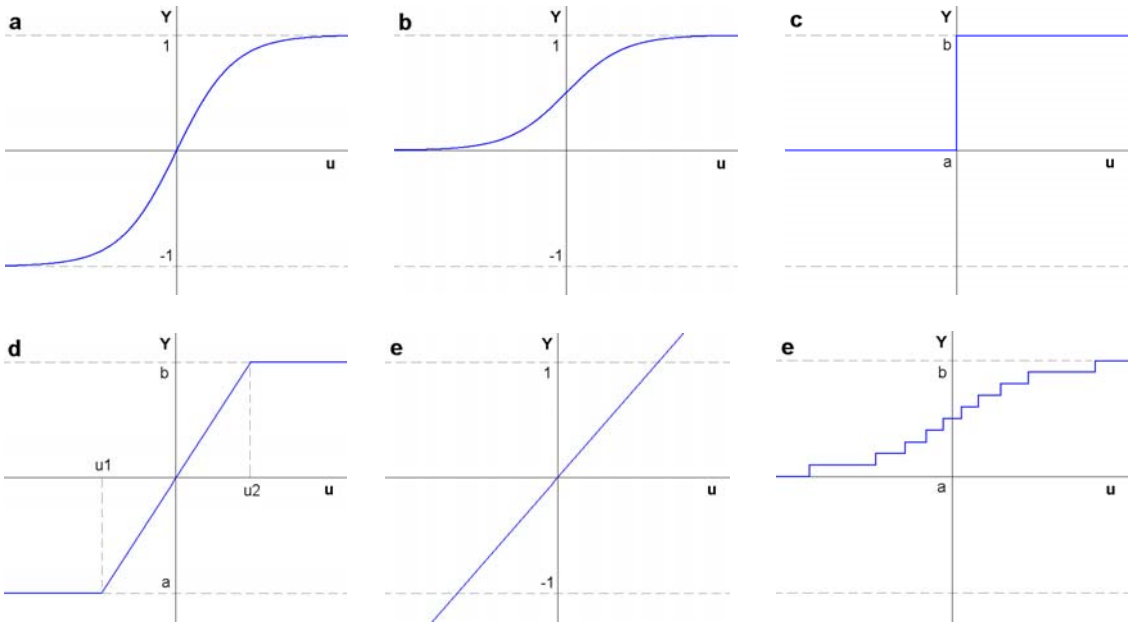

Fig. 1.2. Commonly used neural transfer functions: a – hyperbolic tangent, b – logistic sigmoid, c – threshold, d – linear saturated (semi-linear), f – linear, d – staircase.

The output of a neuron denoted by Y is related to the output of the net function u by a transformation called activation (or transfer) function. Virtually any continuous non-linear and monotone function can be used as neural transfer function [Duch 1999b]. Moreover, if analytical gradient-based methods are used for network training, the functions must be differentiable. The transfer functions most commonly used for multilayer perceptron are summarized in Table 1.1. and their characteristics are shown in Fig.1.2.

1.1.2. Multilayer Perceptron Model

A single layer perceptron is able to classify only linearly separable data. For example, it is not able to solve the Xor problem. This fact was noticed by Minsky and Papert [Minsky 1969] in their famous book "Perceptrons" in 1969. The book contributed to stagnation in research on neural networks for certain time. It was known that multilayer perceptron would solve linearly nonseparable problems, however efficient algorithms for training of MLPs were not known at that time. The first successful algorithm, called backpropagation, was developed several years later [Werbos 1974][Rumelhart 1986] and since that time the field of neural networks has been rapidly developing.

A multilayer perceptron (MLP) is a network that consists of usually two or three layers of neurons and of an additional input layer. The input layer is counted by some authors as a separate network layer while by others it is not. In this thesis a three-layer network refers to a network of two layers of neurons based on the McCulloch and Pitts' model and one additional input layer of neurons that only distribute the input signals, as shown in Fig.1.3.

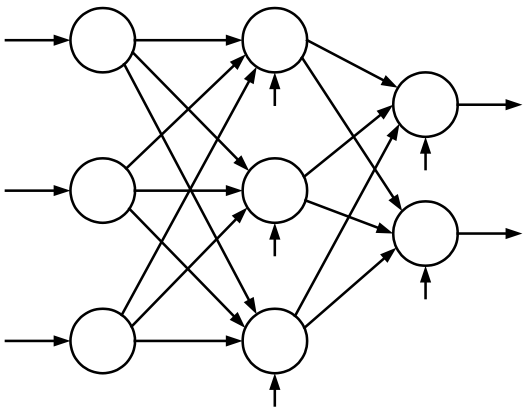

Fig. 1.3. Three-layer fully connected MLP network. Vertical arrows symbolize biases.

In practical implementations there is one input and one output layer and the number of hidden layers can be zero, one or two. During the training process the weights of the output layer and of all hidden layers are optimized. Two successive layers may but do not have to be fully connected. In addition, some weights that prove useless can be removed during or after the network learning process. An MLP network is said to be fully connected if every node in a given layer is connected to every node in the following layer. In some network architectures additional, so called "crossover" connections may be used that directly connect the input layer with the output layer (Fig.1.4).

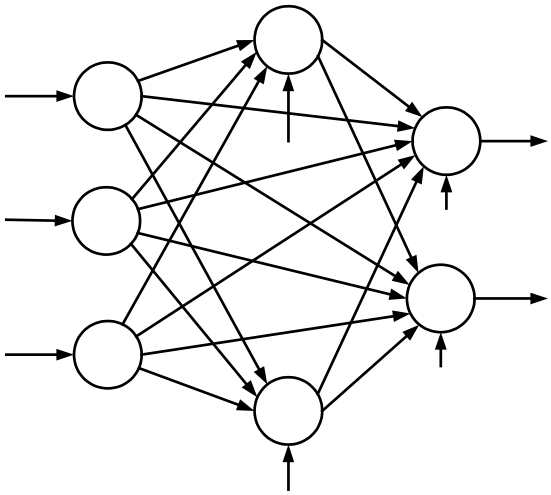

Fig. 1.4. Three-layer MLP network with crossover connections.

1.1.3. Data Classification with Multilayer Perceptrons

Classification is one of the most frequently encountered decision making tasks of human activity. A classification problem occurs when an object needs to be assigned into a predefined class (group) based on a number of observed features (attributes). [Zhang 2000]. Neural networks have emerged as an important tool for classification.

The datasets used by neural networks can be organized in the form of two-dimensional matrices. Each raw of the data matrix contains values of all features that describe a single point in the feature space, called a vector. Each vector is labeled with class information. Thus, the rows of the data matrix contain vectors and the columns contain features. A sample dataset organized in the matrix form is shown in Fig.1.5

Feature 1	Feature 2	Class
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 1.5. Representation of a sample dataset with class labels.

The features can take numerical continuous, numerical discrete or symbolic values (e.g. red, yellow, green). Since MLP networks require numerical inputs, symbolic features must be represented by their numerical counterparts. There are two possible representations. In the first one, each symbolic value is assigned a numeric value and only one input neuron is used for a symbolic feature. In the second one, used in this work, each symbolic feature is represented by a vector of zeros and ones. The length of that vector equals the number of values that the feature can take. All positions in that vector are filled with zeros, except the position corresponding to the actual value of the feature, which takes the value of one. One

input neuron is created for each possible value of the feature, as shown in Fig.1.6. To reduce the number of inputs, one feature value can be considered as default. The default value does not require a corresponding input neuron – if it occurs in the data vector, than no signal is given to any input neuron. Using default values is especially convenient if there are only two discrete or symbolic values in a given feature.

The data classification process consists of two phases. In the training phase the network learns to recognize which data vectors belong to given classes. In the test phase the network is required to classify correctly vectors that have not been used in the training phase.

Each output neuron is assigned a priori to one class. Only the output neuron assigned to the same class as the actual data vector should be activated and its signal should equal one. The signals of all other output neurons should be zero. Nevertheless, it is usually sufficient if the appropriate output neuron signal is higher than 0.5 and higher than the signals of all other output neurons. If this condition is satisfied, than we consider a given vector to be classified correctly.

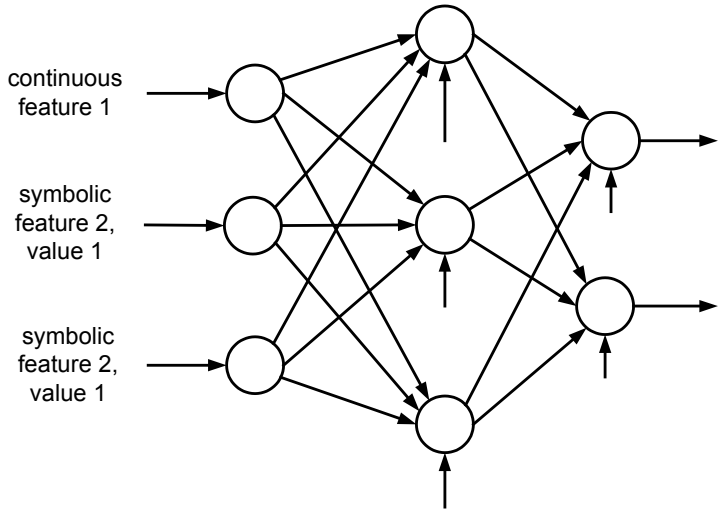

Fig. 1.6. Each value of a symbolic or discrete feature is assigned to a separate input neuron.

Usually before the training phase begins, all weights in the network are assigned a small random values, e.g. within the range (-1;1). Then the training dataset is given to the network inputs vector by vector and the signals propagate through the network. In an ideal situation, only the output neuron assigned to the same class as the actual data vector v is activated and its signal is one, the signals of all other output neurons are zero and the network gives zero error for this vector. In general, the error for a single vector is a function of the differences between the desired and actual signals of all output neurons. The total network error E is the sum of all single vector errors:

$$E = \sum_{v} \sum_{c} f(d_{v,c} - s_{v,c}) \tag{1.2}$$

where d is the desired output signal and s is the observed output signal of the output layer neuron c in response to the training vector v. Many error functions f exist. The most frequently used error function is based on the mean squared error (MSE):

$$E = \sum_{v} \sum_{c} (d_{v,c} - s_{v,c})^2 \tag{1.3}$$

There is some ambiguity in the literature regarding MSE. According to some publications the formula (1.3) represents MSE, while according to other authors the error represented by (1.3) is called sum squared error (SSE) and the average error per single vector in a single output neuron is called MSE (1.4).

$$MSE = \frac{1}{N_v N_c} SSE \tag{1.4}$$

where N_v is the number of vectors in the training set and N_c is the number of output neurons (which usually equals the number of classes, unless there is a default class that does not have a corresponding neuron. No output neuron should be activated in response to the default class vector). However, since MSE is the rescaled SSE, the errors always change proportionally and the mentioned ambiguity practically does not cause any problems.

The aim of the network training is to maximize the classification accuracy as well for the training dataset as for the test dataset. In order to achieve this, the training algorithm minimizes the value of the error function by adjusting values of network parameters. The network error is a function of many parameters, such as the training dataset, network connection structure and weight values. However, if we assume that the training data and network structure is not being changed during the training, the weight values are the only parameters of the error function. The network error function can be imagined as a multidimensional surface, with each weight defining one dimension. Thus, the training algorithms search for a minimum on the error surface.

Except for very simple cases the training algorithms change the weight values iteratively many times. The training set is given to the network inputs vector by vector, the network error is calculated and the weights are adjusted in order to minimize the error. The process of propagating once the entire training set through the network is usually called an "epoch". The process of performing one iteration of the training algorithm is called a "training cycle" (however sometimes it may also be called an "epoch"). Depending on the training algorithm one training cycle can contain a single epoch, several epochs or only a fraction of an epoch.

In supervised learning the network is explicitly told to which class a given vector belongs. By contrast, in unsupervised learning, the network uses unlabeled data (without class information) and has to deduce the classes from data. MLP training algorithms belong to supervised learning methods.

MLP training algorithms can be divided into several categories, such as analytical gradient-based, global optimization or search-based methods. Analytical-gradient based algorithms calculate the derivative of error function with respects to every weight and than change the weights in order to minimize the network error (by moving downwards on the error surface). Global optimization algorithms do not change the weights basing on the gradient direction but search for the minimum in much broader areas. Many methods belong to that group. Search-based methods proposed in this work belong to local methods that instead of analytical gradients use variants of search algorithms. Detailed discussion of MLP training algorithms is presented in the second part of this thesis.

An MLP network used for classification performs a mapping from the input (feature) space to the output (class) space. The aim of the network training is to obtain such weights (and such network structure if it is also modified by the training algorithm) that the mapping reflects the structure of the data and not the single data points. This is known as generalization. The training data frequently contains some noise and the noise should not be reflected in the mapping. If a network generalizes well then it achieves similar classification accuracy on a training set and on a test set. A test contains vectors, which belong to the same data distribution, but which have never been used in the training process.

Often the availability of data is limited and using a part of it as a test set is not practical. An alternative is to use the procedure of crossvalidation. In *k*-fold crossvalidation the training set is randomly divided into *k* subsets, the network is trained using *k*-1 subsets and tested on the remaining subset [Bullinaria 2002]. Typically *k*=10 is considered reasonable. The process of training and testing is then repeated *k* times, using each one a different subset as a test set. The average classification accuracy on the *k* test subsets gives the estimate of the network performance.

1.1.4. Applications of Multilayer Perceptrons

The advantages of neural networks over conventional programming lies in their ability to solve problems that do not have an algorithmic solution or the existing solution is too complex to be found. Problems that were unsolvable using logical systems are now being tackled using an artificial neural network approach [Pennington 2003].

Multilayer perceptron is the most widely used type of neural networks and thousands of applications of MLP networks are known. These problems are in areas as diverse as medical diagnosis [Sordo 2002][Adamczak 2001][Jankowski 1999], medical image recognition [Pincho 1993][Kabarowski 1999][Pennington 2003], time series prediction [Osowski 1996], data compression [Gabriel 2003][Verma 1999], defect detections in materials [Karras 2001], bankruptcy prediction [Altman 1994][Raghupathi 1996], music classification [Maihero 2004], solar collectors sensitivity analysis [Zarate 2004], handwriting recognition [Garris 1998][Lee 1993],viruses and internet worms detection [Bielecki 2004], and many others. The applications found for neural networks continue to grow at a rapid rate.

1.1.5. Further Development of Multilayer Perceptrons

Using neural networks problems can be solved without the need to understand how a solution is achieved. As long as there are a finite number of attributes to the problem and an expected result, neural networks can find a solution to the problem. This makes them a useful tool for anyone working on pattern recognition problems. Nevertheless, many people do not trust neural networks because they do not explain how they have reached the solution. Especially in medicine, where the knowledge of how the result has been obtained is important, many doctors do not want to use neural networks, in spite they have higher diagnosis accuracy than other systems [Sordo 2002]. Although some attempts were made to extract logical rules from trained neural networks, many people still consider them as black boxes [Duch 2001, 2004c].

The aim of this thesis is not only to propose new algorithms for MLP training and logical rule extraction but also to explain, as far as possible, how the networks work. Thus, a great emphasis is placed on the understanding of neural learning processes. Frequently plots are used to show many interesting aspects, including visualization of high-dimensional MLP weight spaces. A better understanding of how the networks work also allows us to develop better algorithms for the network training and logical rule extraction.

1.2. Visualization and Properties of MLP Error Surface

1.2.1. The Purpose of MLP Learning Visualization

Visualization of learning processes in neural networks shows the dynamics of learning, allows for comparison of different network structures and different learning algorithms, displays training vectors around which potential problems may arise, shows differences due to regularization and optimization procedures, investigates stability of network classification under perturbation of original vectors, and allows for estimation of confidence in classification of a given sample.

There are many known methods of high dimensional data visualization [Atkosoft 1997][Naud 2001], however most of them are not suitable for visualization of learning processes in neural networks. Thus, several methods especially dedicated to MLP learning have been proposed in the literature. In a Hinton diagram [Hinton 1986] each weight value in the network is represented by a box. The size of the box gives the magnitude of the weight, whereas the color (e.g. white or black) indicates whether the weight is positive or negative. The Bond diagram [Wejchert 1991] visualizes the weights on the topology of the network. Units are represented as simple points, with “bonds” of varying length (weight magnitude) and color (weight sign) emanating from unit outputs towards other units. Wejchert and Tesauro [Wejchert 1991] also consider a trajectory diagram, which emphasizes the visualization of the learning process itself by representing the multidimensional coordinate system in a two-dimensional plane by a star-like projection. The projection allows weight vectors to be plotted radially component by component, but it is practically limited to about six weights in the network. The plots of two different weight values against the error function, which produce a two-dimensional slice of the n-dimensional error surface, have also been used in the literature [Gallagher 2000].

PCA (Principal Component Analysis) was used for three-dimensional visualization of backpropagation learning trajectories [Gallagher 2000, 2003], for visualization of learning trajectories of several training algorithms [Kordos 2004b, 2004c, 2005] and for visualization of MLP error surfaces [Kordos 2004a, 2004c]. Visualization of each layer neuron signals was considered in [Duch 2004a]. The dependencies between the gradient components and the error surface sections in particular directions [Kordos 2004d, 2005] and the changes of weight values can also provide information that can be practically used to tune some training methods.

The most interesting visualization methods together with several statistics from network trainings are presented in the following chapters. The purpose of that visualization is to enhance the understanding of neural network processes and to give some hints for training algorithms design and optimization. The practical conclusions from the study allow for shortening training times and increasing the stability and accuracy of network learning processes. In this part of the thesis, as well by "epoch" as by "training cycle" we will understand one iteration of the training algorithm, after which all the weights change their values.

1.2.2. MLP Error Surface

The error surface (ES) $E(\boldsymbol{W})=\sum_X||\boldsymbol{Y}-\boldsymbol{M}(\boldsymbol{X};\boldsymbol{W})||$ of a neural network is defined in the weight space $\boldsymbol{W}$ (including biases as W_0 weights) for a given set of training vectors $\boldsymbol{X}$, desired output vector $\boldsymbol{Y}$ and a vector mapping $\boldsymbol{M}(\boldsymbol{X};\boldsymbol{W})$ provided by the neural network. Only the multilayer perceptron (MLP) networks are considered here. Probably it would be possible to use similar techniques to investigate other types of feedforward networks, however it has not been attempted yet. An MLP training process can be defined as a search for a global minimum on the hyper-surface $E(\boldsymbol{W})$, where it creates a learning trajectory.

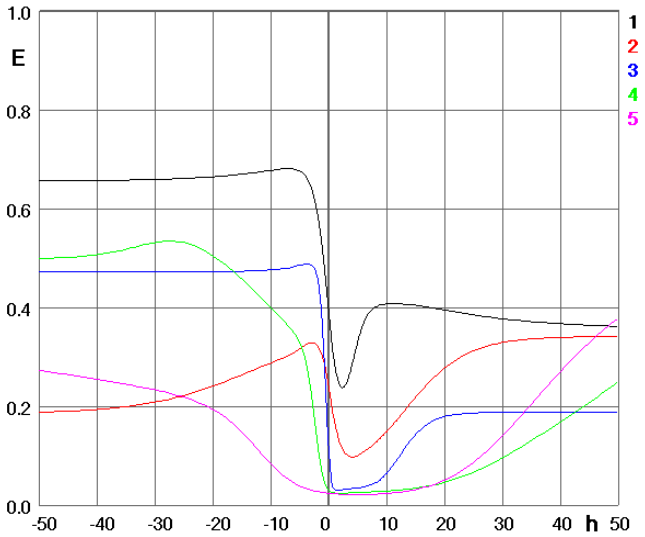

Fig. 1.7. MLP error surface sections of Iris (4-4-3) in gradient directions obtained using numerical gradient training cycles 1÷5.

One way to understand better the learning dynamics of MLPs is to visualize both the ES and the learning trajectory using projections of the original space into a two- or three-dimensional subspace. The projection directions should preserve most information about the original surface. In two-dimensional visualizations, the error value is displayed on the vertical axis, and one direction in the weight space on the horizontal axis. A good choice is either the local gradient direction or the first principal component direction that is calculated in the weight space.

A sample plot showing the change of the mean squared error (MSE) in the gradient direction is shown in Fig.1.7. The training of an MLP with a single hidden layer composed of four nodes has been done on the Iris data, frequently used for illustrations (chapter 1.2.5.1). The numbers of neurons in successive layers are given in brackets after the dataset name. For example (6-4-3-2) means that the network has 6 inputs, 4 neurons in the first hidden layer, 3 neurons in the second hidden layer and 2 neurons in the output layer. The lines in Fig.1.7. were created by changing the length of the weight vector $\boldsymbol{W}$ in the gradient direction h. The starting point (h=0) for each line is in the minimum found along the previous training cycle gradient direction. The first curve has a narrow and deep minimum, indicating that a rather narrow funnel is traversed on the error surface. The second and the subsequent curves reach lower error levels and are broader, indicating that a broad plateau has been reached. This should be expected in all problems where separation of different categories is relatively easy and the error surface should be insensitive to weight changes corresponding to rotations and shifting of decision borders that do not affect the separation.

It seems worthwhile to investigate the error surfaces not only in two, but also in three-dimensional spaces. PCA (Principal Component Analysis) is a natural choice for visualizing the weight space because it provides components from which the original weight space may be reconstructed with the highest accuracy.

Fig.1.8-left shows the error surface projection into two principal components c1 and c2, which has been obtained using weights from the same network training, as the error surface sections shown in Fig. 1.7. The learning trajectory lies on the bottom of one of the ravines. Beginning the training from another starting point could result with the trajectory lying on the bottom of another ravine. Learning trajectories will be discussed in chapters 1.3, 2.3.9 and 2.4.4.

1.2.3. Research Methodology

1.2.3.1. Overview of Research Methodology

In order to visualize the error surface the following procedure is used:

1. A network is trained using either standard backpropagation [Rumelhart 1986][Hen 2002][Bullinaria 2002], Levenberg-Marquardt second-order algorithm [Ranganathan 2004][Marquardt 1963], scaled conjugate gradient [Moller 1993], numerical gradient [Kordos 2003b], the simplest search-based method that changes one weight at a time [Kordos 2003a] or its modified version with variable step search [Kordos 2004b]. It is worth to remark now that the experimental results do not depend significantly on the training algorithm.
2. Weight vectors $\boldsymbol{W}(t)$ after each training cycle t are collected into the weight matrix $\boldsymbol{WM}$.
3. PCA (Principal Component Analysis) is performed on the weight covariance matrix (the covariance matrix of the weight matrix).
4. Three-dimensional error surface projections are plotted. The horizontal axes correspond to the first and second PCA direction and the vertical axis shows the network error value.

1.2.3.2. Principal Component Analysis

Principal Component Analysis (PCA) is a technique that reduces the data dimensionality while preserving as much of the high dimensional space properties as possible. PCA is performed by a rotation of the original high dimensional coordinate system and then discarding the axes along which the data has the smallest variance. The rotation is done in such a way that the variances along the successive axes decrease as quickly as possible.

Each weight vector $\boldsymbol{W}(t)=[w_{1t},\ldots,w_{nt}]$ is defined by a single point in the weight space. The training produces a set of points, on which PCA can be performed. Weight vectors after each training cycle t are collected into the weight matrix $\boldsymbol{WM}$:

$$\boldsymbol{WM} = \begin{bmatrix} \boldsymbol{W}(0) \\ \vdots \\ \boldsymbol{W}(T) \end{bmatrix} = \begin{bmatrix} w_{10} & \ldots & w_{n0} \\ \vdots & \ddots & \vdots \\ w_{1T} & \ldots & w_{nT} \end{bmatrix} \tag{1.5}$$

where T is the number of training cycles. PCA can be performed either directly on the weight matrix $\boldsymbol{WM}$ or on the weight covariance matrix $\boldsymbol{CM}$:

$$\boldsymbol{CM} = \begin{bmatrix} c_{11} & \ldots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \ldots & c_{nn} \end{bmatrix} \tag{1.6}$$

The covariance matrix is a symmetric matrix, its entries c_{ij} are calculated as

$$c_{ij} = \frac{1}{T}\sum_{t=0}^{T}\left(w_{it} - \overline{w_i}\right)\left(w_{jt} - \overline{w_j}\right) \tag{1.7}$$

and they represent the covariance between the weight w_i and w_j, where T is the number of training cycles, n is the number of network weights and the mean weight value is calculated as

$$\overline{w_i} = \frac{1}{T+1}\sum_{t=0}^{T} w_{it} \tag{1.8}$$

The eigenvectors v_i and their corresponding eigenenvalues λ_i of the weight covariance matrix are the solution of the characteristic equation:

$$\boldsymbol{CM}\cdot v_i = \lambda_i \cdot v_i, \ \ for\ i=1,\ldots,n \tag{1.9}$$

If the weight vector has n components, the characteristic equation becomes of order n. This is easy to solve only if n is small. Solving eigenvalues and corresponding eigenvectors is a non-trivial task, and many methods exist. One way to solve the eigenvalue problem is to use a procedure called singular value decomposition (SVD) [Kalman 2001]. The SVD procedure presented in "Numerical Recipes in C" [Press 1992] was used in the calculations. By ordering

the eigenvectors in the order of descending eigenvalues (largest first), we can create an ordered orthogonal basis with the first eigenvector having the direction of largest variance of the data [Hollmen 1996]. The data variance in a given eigenvector direction is proportional to the eigenvalue corresponding to the eigenvector. In this way, the directions in which the data has the most significant information can be found. Let $\boldsymbol{V}$ be a matrix consisting of eigenvectors of the covariance matrix as the row vectors. By transforming the weight vector $\boldsymbol{W(t)}$, we get

$$\boldsymbol{Y} = \boldsymbol{V} \cdot \left(\boldsymbol{W}(t) - \overline{\boldsymbol{W}} \right) \tag{1.10}$$

which is a point in the orthogonal coordinate system defined by the eigenvectors. Thus, the axes of the new coordinate system are in the eigenvector directions. Components of $\boldsymbol{Y}$ can be seen as the coordinates in the orthogonal basis. We can reconstruct the original weight vector $\boldsymbol{W(t)}$ from $\boldsymbol{Y}$ by

$$\boldsymbol{W}(t) = \boldsymbol{V}^T \cdot \boldsymbol{Y} + \overline{\boldsymbol{W}} \tag{1.11}$$

using the property of an orthogonal matrix $\boldsymbol{V}^1 = \boldsymbol{V}^T$. The data variance in each eigenvector direction, which will be further called simply the first, second and so on PCA direction is proportional to its corresponding eigenvalue. Only some directions with the greatest variance are preserved and all remaining directions are discarded.

SVD can be calculated either on the weight matrix or on the weight covariance matrix. The resulting plots are of similar nature, although the eigenvalue distribution is different. A weight matrix gives a smaller first to second eigenvalue ratio and bigger the least significant eigenvalues, but in both cases the first and second PCA directions typically contain about 95÷97% of the total variance. Nevertheless, SVD on the covariance matrix has a significant advantage: the error surface projections obtained in the experiments differ less from training to training (are less influenced by the random initial distribution of weights). For this reason all plots presented here are based on SVD on the covariance matrix, except for the two sample ES presented in Fig.1.11.

1.2.3.3. Plot Construction

Vertical axis in the plots shows the relative error $E=E(\boldsymbol{W})/N_V N_C$, where N_V is the number of vectors and N_C is the number of classes in the training set. For all error functions based on Minkovsky's metric $\|\cdot\|$ the error function is bounded from above by $N_V N_C$, thus the relative error E is bounded by 1. Horizontal axes show distances in the weight space in c_1 and c_2 PCA directions corresponding to the first and second eigenvector of the weight covariance matrix. Thus in a given point (c_1,c_2) of the plot the network weight vector $\boldsymbol{W}(c_1,c_2)$ is determined by the following equation:

$$\boldsymbol{W}(c_1, c_2) = \boldsymbol{W}_0 + c_1 v_1 + c_2 v_2 \tag{1.12}$$

where v_1 is the first and v_2 is the second eigenvector of the weight covariance matrix, c_1 and c_2 are the distance along the horizontal axes and $\boldsymbol{W_0}$ is the vector of constant weights. In most of the plots $\boldsymbol{W_0}$ consists of zero weights for the simplicity reason because $\boldsymbol{W_0}$ containing the mean weight values during the training produces plots that look very similar and that are only horizontally shifted. The aim here is to find the most interesting projection directions. The equation (1.11) refers to the data from the weight matrix. When the plot is drawn it uses only the PCA-based directions, but particular points on the error surface are not present in the weight matrix, thus the equation (1.12) as the generalized version of (1.11) is used for error surfaces. However, the equation (1.11) always applies to the visualization of learning trajectories in the PCA-based directions. Non-zero $\boldsymbol{W_0}$ vectors are considered in chapters 1.2.10.2 and 1.3.

The character of ES is determined by the dataset and network structure. In the experiments MLP networks were trained for data classification for as many training cycles as were required to bring them close to convergence. There was not a strict stopping criterion, since the results were very little sensitive to the stopping point, but in most cases the trainings were stopped when the error decrease reached about 95% of the possible error decrease. Sometimes the stopping point was intentionally determined in another way in order to show some phenomena, but this will be mentioned explicitly. The number of epochs varied depending on a training algorithm and a dataset. At the final training stage weights of output neurons tend to grow quicker then those of hidden neurons, but since the training was stopped before convergence, weights of each layer had still comparable contributions in determining PCA directions. The training was repeated several times for a given method with various random initial weights.

Neither the random weight distribution nor the training method has significant influence on the shape of ES presented in the space of the two main PCA components. The projection of error surface for a given dataset and network structure may differ a bit - it may rotate from one plot to another, its elements may be a bit higher or lower, but the overall structure is well preserved.

To obtain the most reliable ES projections, PCA should be calculated using the weight matrix containing data from the training cycles ranging from the initial weights (from the starting point) to that point when the error begins to change very slowly. Otherwise, especially if the initial training cycles with rapid error changes are omitted, some distortion described in later chapters will appear.

In most of the plots presented here logistic sigmoids are used as neural transfer functions but ES projections obtained with hyperbolic tangent do not differ significantly. Also some examples of ES obtained with other types of transfer functions will be presented.

Over 20 datasets were used in the experiments, about half of them comes from the UCI machine learning database repository [Mertz 1998]. To be concise only one ES typical for a given situation will be shown; the others are qualitatively similar.

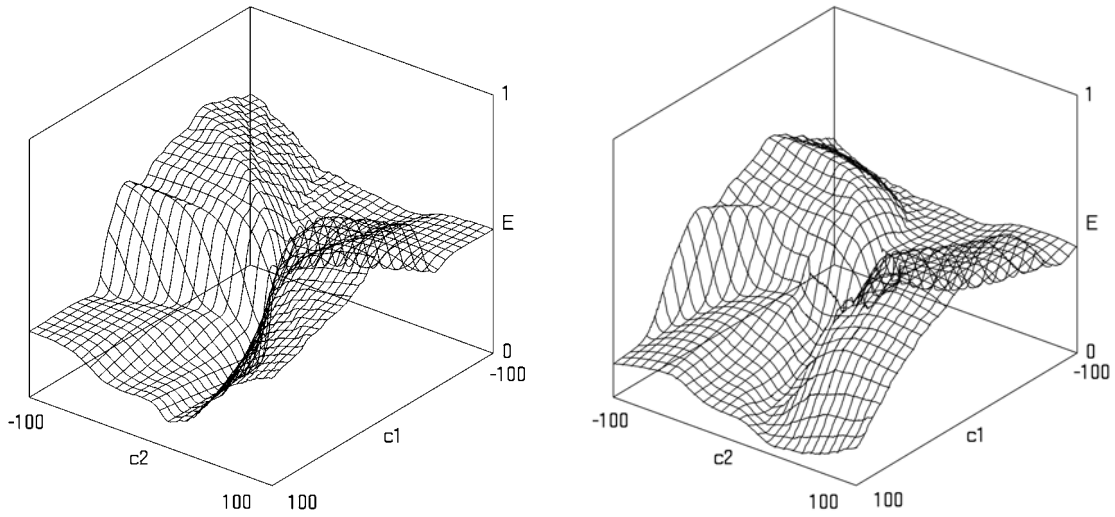

Fig. 1.8. Left: MLP error surface of Iris (4-4-3) displayed in two PCA directions, the plot was made using the same data as in Fig.1.7. Right: MLP error surface of Iris (4-4-3) showing more faithfully how the ES might look like.

Although PCA projections seem to be very good for ES visualization they do not reveal certain aspects of the original ES. The detailed reasons for this will be discussed in later chapters. At this stage three major differences between the original ES and their PCA projections are worth pointing out:

- The ravines in which the training trajectories lie are curved, not straight as shown in the PCA projections.
- The original ravines tend to be steeper (starting higher and ending lower) than those shown in PCA projections.
- Sometimes shallow local minima close to the ES center are visible in PCA projections, although they do not exist in the original ES.

Fig. 1.8-right is a modified version of fig.1.8-left that shows how the real ES might look like, addressing the points mentioned above. It can be only imagined or visualized if the projection directions are different in different fragments of the plot, however the detailed approaches to such a visualization model have not been attempted yet.

Typically the first and second PCA directions contain together about 95% of the total variance and therefore, despite of the three shortcomings mentioned above, the plots reflect ES properties quite well. There is a strong correlation between the growth of a given weight

during the training *growth*(*w*) and its corresponding entry in the first principal component "*1st PC(W)"* (in the first eigenvector of the weight covariance matrix) (Fig. 1.9-left). The entries in the further principal component vectors seem to be uncorrelated with value of growth of their correspondent weights (Fig. 1.9-right).

Table 1.1. Eigenvalues and variance captured by the *PC*-th PCA component for the same training as in Fig.1.7 and 1.8.

PC	1	2	3	4	5	6	7	8	9	10
eigenvalue	33.204	1.4550	0.5969	0.2554	0.1578	0.0679	0.0547	0.0324	0.0265	0.0191
% current variance	0.9245	0.0405	0.0166	0.0071	0.0044	0.0019	0.0015	0.0009	0.0007	0.0005
% total variance	0.9245	0.9651	0.9817	0.9888	0.9932	0.9951	0.9966	0.9975	0.9982	0.9988

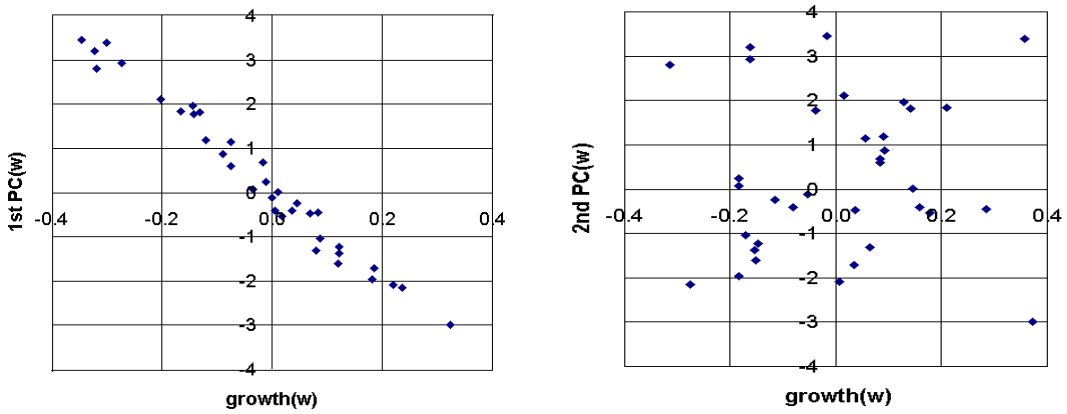

Fig. 1.9. Left: Correlation between a given weight entry in the first eigenvector of the weight covariance matrix *1st PC(W)* and the weight growth during the training *growth(w)* of Iris (4-4-3). Right: Correlation of the *2nd PC(W)* and *growth(w)* for the same training as in Fig. 1.7 and 1.8.

ES plots are based on weight matrices containing the weights from network trainings, which minimize the network error. Thus, the trajectories traverse rather the parts of the weight space with lower error values than the parts with higher error values. As a result, we can see the projected ES rather in the bottom than in the top part of the cube. It is not recommended to try to traverse and display a more diverse area of the weight space by combining the weights from several trainings into one weight matrix because the average value of each weight in such a matrix tends to zero as the number of trainings grow, as a result the ES projection approaches a horizontal plane.

1.2.3.4. Independent Component Analysis

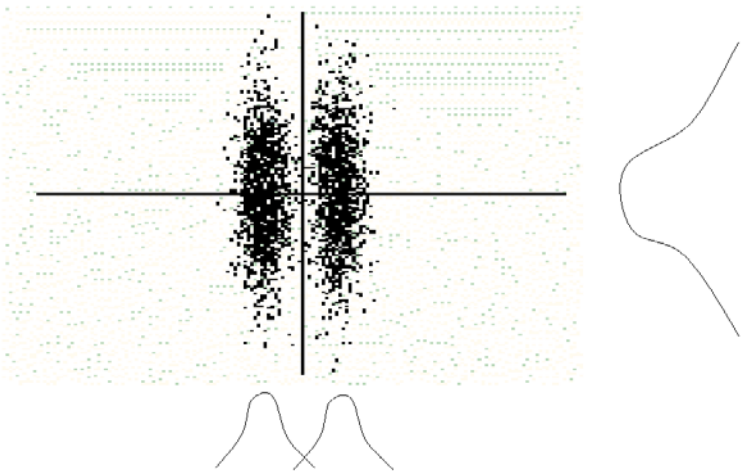

Fig. 1.10. The data in this figure is clearly divided into two clusters. However, the principal component, i.e. the direction of maximum variance, would be vertical, providing no separation between the clusters. In contrast, the strongly nongaussian independent component direction is horizontal, providing optimal separation of the clusters. (the figure comes from www.cis.hut.fi/aapo/papers/NCS99web/node8.html)

PCA projections are in the directions of maximum variance, thus even if the data is clearly divided into two clusters, PCA may not reveal this structure. ICA (Independent Component Analysis) projections are in the maximally nongaussian directions, providing usually good separation of clusters, though not necessarily the directions of maximum variance [Leino 2004]. So the ICA-based approach may show some additional ES properties, not visible in PCA projections, and produce generally more complex ES projections with more details.

ICA starts with a vector of observations $\boldsymbol{x}$ (frequently PCA is used as data preprocessing for ICA and $\boldsymbol{x}$ is then the original vector projected into the PCA directions):

$$\mathbf{x} = (x_1, ..., x_n) \tag{1.13}$$

The basic assumption here is that each of these observations can be derived from a set of n independent components:

$$\mathbf{x}_i = a_{1i}s_1 + ... + a_{in}s_n \tag{1.14}$$

or, using a matrix notation, $\boldsymbol{x}=\boldsymbol{A}\boldsymbol{s}$. Here $\boldsymbol{s}=(s_1,...,s_n)$ is a random vector – the latent variables, or independent components, and $\boldsymbol{A}$ is a $m \times n$ mixing matrix. The task of ICA is to find both **s** and **A**. However, the matrix $\boldsymbol{W}=\boldsymbol{A}^{-1}$ is directly searched for, so that the sources $\boldsymbol{s}=\boldsymbol{W}\boldsymbol{x}$ can be estimated from vector $\boldsymbol{x}$ of the observed signals by optimizing a statistical independence criterion. The basic assumption of ICA is that the components s_i are independent of each other, that is $P(s_i,s_j)=P(s_i)P(s_j)$.

The entropy H of a random vector $\boldsymbol{x}$ of density $p_x(u)$ is defined as

$$H(p_x) = -\int p_x(u) \log p_x(u) du \tag{1.15}$$

$H(p_x)$ is maximal for a gaussian random vector x. The negentropy J is defined by the difference of entropy between $\boldsymbol{x}$ and a gaussian random vector $\boldsymbol{x_g}$ of the same covariance matrix as $\boldsymbol{x}$:

$$J(x) = H(p_{xg}) - H(p_x) \tag{1.16}$$

The FastICA algorithm [Hyvarinen 1999, 2001] uses the following estimation of negentropy:

$$J(x) = \{E[g(x)] - E[g(x_g)]\}^2 \tag{1.17}$$

where g is any non-quadratic function. The maximization of the measure of negentropy is done by an iteration scheme, which for one independent component w is:
1. choose an initial (e.g.) random weight vector w
2. $w^+ \leftarrow E\{x\, g(w^T x)\} - E\{g'(w^T x)\} w$, with $g(u) = tanh(u)$, or $g(u) = u\, exp(-u^2/2)$
3. $w \leftarrow w^+ / ||w^+||$
4. if not converged (i.e. if old and new w point in different directions), go to 2

The algorithm can be run for each independent component i. To prevent different vectors $\boldsymbol{wx}$ from converging in the same direction, the vectors are decorrelated after every iteration, using for example the decorrelation of matrix W:

$$\boldsymbol{W} = (\boldsymbol{W}\, \boldsymbol{W}^T)^{-1/2}\, \boldsymbol{W} \tag{1.18}$$

The FastICA algorithm was used in the calculations, resulting in very similar projections to those obtained with PCA on the covariance matrix. The global character of both projections is the same, only some more details are visible in ICA projections, mainly as the folded ridges (Fig.1.11-left).

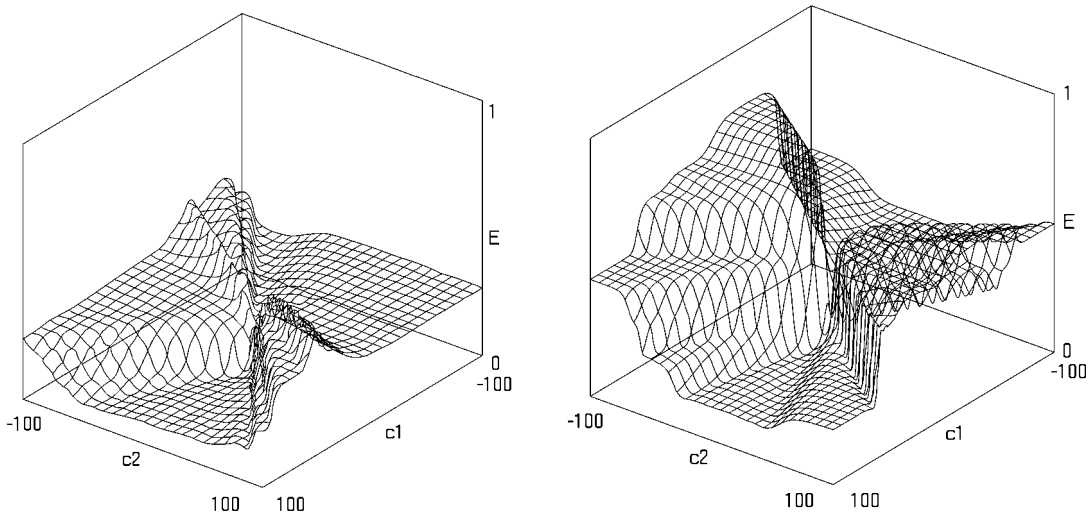

Fig. 1.11. A comparison of Iris (4-4-3) error surface projection in ICA directions (left) and PCA directions calculated by SVD on the weight matrix (right) for the same training as in Fig.1.7 and 1.8.

The first ICA direction is almost parallel to the first PCA direction with the cosine between them about 0.99, but the second directions seem uncorrelated with the cosine between them usually below 0.3. Change of various FastICA algorithm parameters (e.g. the function *g*) did not noticeably change the plots. Generally, the hopes to see much more details that would reveal some more ES aspects using ICA-based projections were disappointed. Thus, only one plot obtained with an ICA-based projection is presented in this thesis for comparison (Fig. 1.11-left) and all further plots are shown in PCA-based projections.

1.2.3.5. Two-weight Coordinate System

Coordinate systems based on any two-weight directions do not provide so much information as PCA systems. A large number of error surface projections of networks with more than 10÷20 weights are composed of four horizontal planes, which are sometimes reduced to two or even a single plane. The surfaces have similar characters for many datasets and network architectures and resemble the ES projection shown in Fig. 1.12-left. More complex shapes of ES projection in two-weight systems are rare for medium to large networks.

In networks with significantly more hidden neurons then the number required to learn the task, the neurons perform highly redundant roles. In that case changing any two weights of the trained network does not change the error because then signals propagate through the redundant paths and ES in a two-weight system creates only one horizontal plane.

1.2.4. Network Structure Influence on Error Surface

Networks without hidden layers have very simple ES consisting only of some horizontal or slightly inclined half-planes, situated on various heights, with slopes connecting them (Fig.1.12-left).

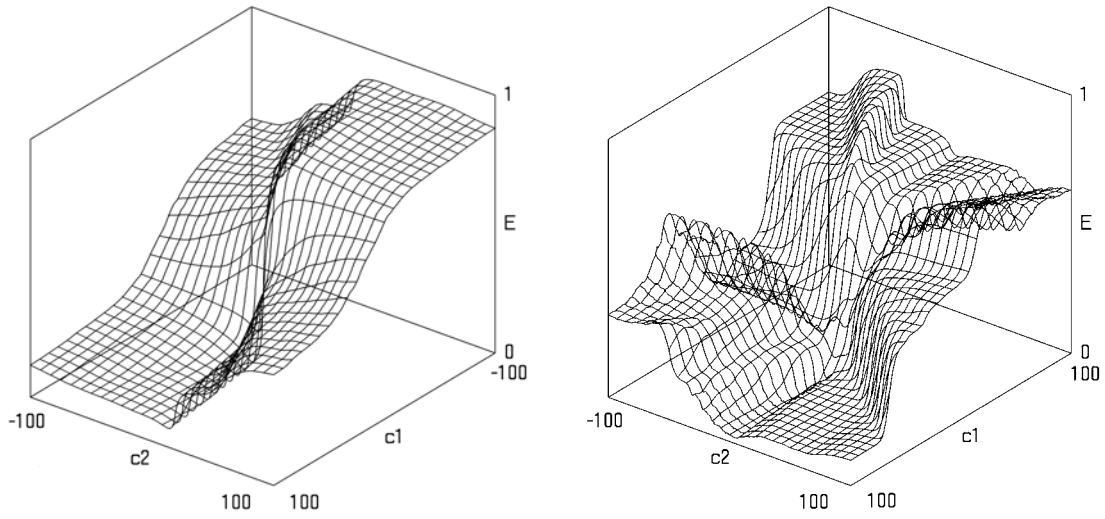

Fig. 1.12. Left: ES of a 2-layer network (Iris 4-3). Right: ES of a 4-layer network (Iris 4-4-4-3).

ES of networks with hidden layers has a starfish structure. An interesting depiction of it was given by Denker et. al. [Denker 1987] "E(W) surface resembles a sombrero that has been warped in certain symmetric ways: near the middle (w=0) all configurations have moderately bad E values. Radiating out from the center are a great number of ridges and valleys. The valleys get deeper as they go out, but asymptotically level out. In the best valleys, E is exactly or asymptotically zero, other valleys have higher floors". Pictures presented in this thesis confirm that global minima rarely create craters but frequently ravines reaching their minimum in infinity. This corresponds to the infinite growth of (usually output layer) weights when continuing the training enough long.

Each of *h* hidden neurons may be labeled with an arbitrary and unique number from *1* to *h*. Renumerating the network parameters does not change the mapping implemented by the network, thus giving *h*! permutational symmetries. A neural activation function for which *f(-x)=-f(x)+const* gives further 2^h sign-flip symmetries [Sussmann 1992]. This gives together $2^h h!$ equivalent global minima. A training algorithm converges to that minimum which is the easiest to reach from the starting point. Only some of the minima are clearly visible in the PCA projections.

Four layer networks have more complex ES than the three layer ones, even with fewer neurons. Thus they can map more complex data (Fig.1.12-right). In 3-layer networks with crossover connections (Fig. 1.4) the output layer is connected directly to both: the input (as in 2-layer networks) and hidden layer (as in 3-layer networks). Consequently their ES displays features of 2-layer networks (low symmetry of ES) and 3-layers networks (complexity of ES) (Fig.1.13-left).

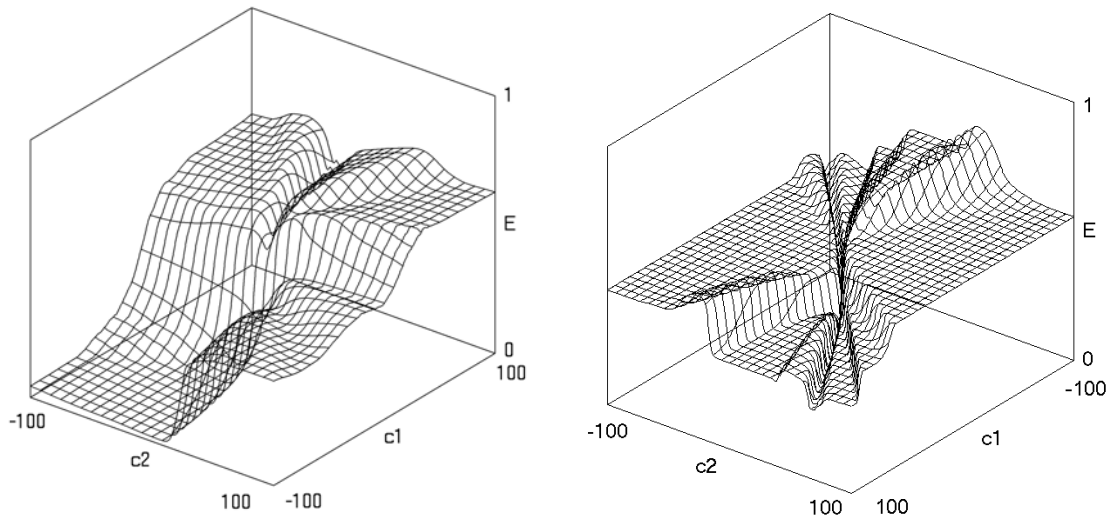

Fig. 1.13. Left: ES of a 3-layer network with crossover connections (Iris 4-4-3). Right: ES of a 3-layer network with too many hidden neurons (Iris 4-100-3).

Too few neurons in any hidden layer make a bottleneck and the network cannot learn the task. The ES consists of some horizontal planes all placed relatively high with some disturbances between them, but does not contain the characteristic ravines leading to global minima (not shown here).

The number of global minima visible in PCA projections initially grows when the number of hidden neurons increases, but with too many hidden neurons big horizontal planes begin to appear (Fig.1.13-right). This effect caused by the weight redundancy is visible more clearly in two-weight coordinate systems, where the projected ES is almost flat since many weights must be changed at the same time to change the error.

1.2.5. Training Dataset Influence on Error Surface

1.2.5.1. Description of the datasets used in experiments

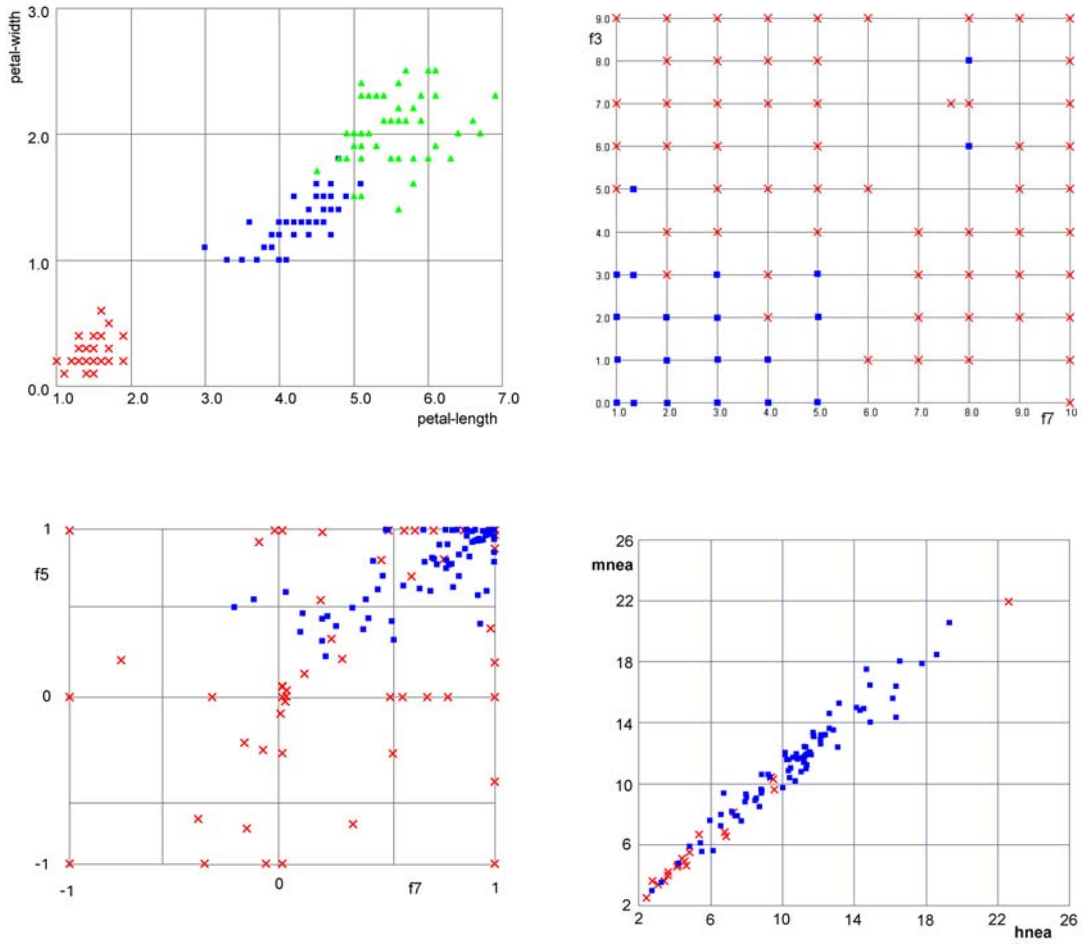

Fig. 1.14. The distribution of class instances shown in the space of two most informative features. Left-top: Iris, right-top: Breast, left-bottom: Ionosphere, right-bottom: Appendicitis.

1. Iris (Fig.1.14.left-top): 4 continuous features (sepal-length, sepal-width, petal-length, petal-width), 3 classes, 150 vectors, 50 in each class. Two of the features (petal-length, petal-width) are most informative for classification, the remaining two features are more noisy and do not provide additional information. Although the classes are well separated, three

classes make the training a bit longer than the training of the Breast dataset. The accuracy that may be achieved in 10-fold crossvalidation is about 96%. The dataset is publicly available at UCI [Mertz 1998].

2. Wisconsin Breast Cancer (Fig. 1.14.right-top): 10 continuous features (f1,...,f10), 2 classes (class 1-red cross in , class 2-blue square), 699 vectors, 458 in class 1 and 241 in class 2. The classes are separated rather well, the set is very easy for training. The possible accuracy in 10-fold crossvalidation is about 96%. The dataset is publicly available at UCI [Mertz 1998] and described in chapter 3.2.12.4.

3. Ionosphere – training dataset (Fig. 1.14.left-bottom): 34 continuous features (f1,...,f34), 2 classes , 200 vectors, 100 in class 'good' and 100 in class 'bad'. The classes are not so well separated as in the two first datasets. The possible accuracy in 10-fold crossvalidation is about 94%. The dataset is publicly available at UCI [Mertz 1998].

4. Appendicitis: 10 continuous features (f1,...,f10), 2 classes (class 1-red cross in Fig. 1.14.right-bottom, class 2-blue square), 106 vectors, 21 in class 1 and 85 in class 2 (strongly asymmetric class distribution). The classes are not so well separated as in the two first datasets. The possible accuracy in 10-fold crossvalidation is about 89%. The dataset is described in chapter 3.2.12.3.

1.2.5.2. Experimental Results

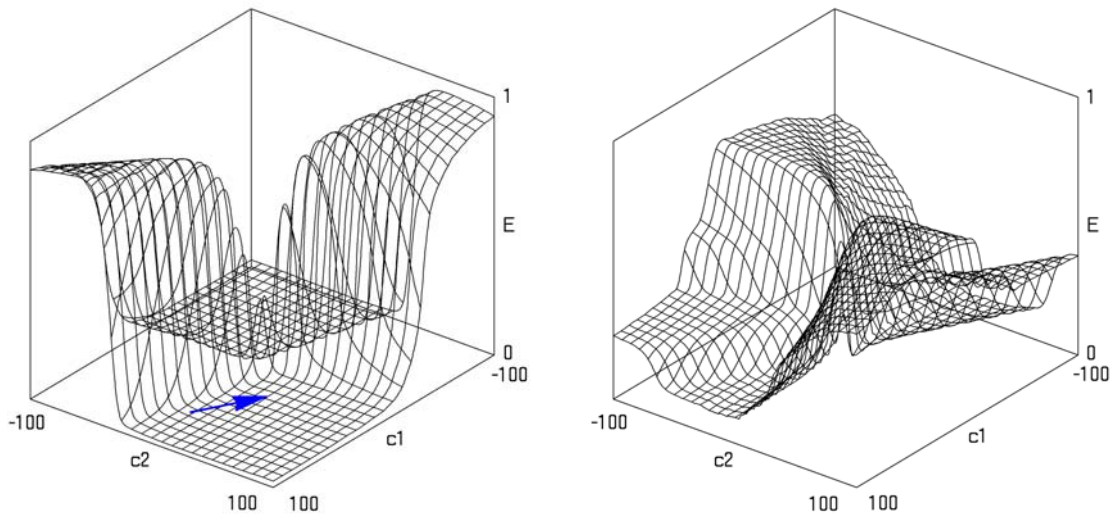

Fig. 1.15. Left: ES of Breast (10-4-2) The arrow shows a point to which the jump described in chapter 1.2.10.2 was made . Right: ES of Ionosphere (34-4-2).

A similar network structure x-4-2 has been used for various datasets. Generally the following tendencies can be observed:

- More complex training datasets produce more complex ES with more ravines, especially for data that is not approximately linearly separable.
- Equal classes of examples lead to a more symmetric ES.

Breast (Fig.1.15-left) has two classes with few overlapping vectors and therefore the simplest ES. Iris (Fig.1.8-left) has 3 classes with little overlap and Ionosphere (Fig.1.15-right) 2 classes with more overlap – they both give similar ES.

Appendicitis (21 vectors of class 1 and 85 of class 2) gives a highly non-symmetric ES (Fig.1.16-left). Setting the network weights (chapter 1.2.9) to the values represented by the appropriate parts of the error surface indicates that the big flat area situated in the front part of the plot corresponds to the majority classification accuracy (for the points located on this fragment of ES the predicted class is class 2). Frequently training of datasets with unbalanced classes is more difficult because this part of ES is very flat and very broad. It is easy to get there, but difficult to leave this area. The ravines between this part and the higher situated areas in the back of the plot correspond to the optimal classification accuracy (about 90-92% in the case of Appendicitis). But the same dataset with only 42 vectors left (all of class 1 and randomly chosen 21 vectors of class 2) produces a quite symmetric ES (Fig.1.16-right). The topic of unbalanced classes will be further discussed in chapter 1.6.

An *n*-bit parity is a problem, where the dataset has *n* features and two classes. Each of the features can take two values: zero or one. If an even number of features in a given vector take the value of one then the vector belongs to the first class, otherwise it belongs to the second class. Xor, which is a 2-bit parity problem, is linearly non-separable and therefore has a complex ES (Fig.1.17-left). 6-bit parity is linearly non-separable and has 32 clusters per class (Xor has only two) and its ES is very intricate, however symmetric because the number of vectors in each class is equal (Fig.1.17-right). Moreover, datasets that are easier for training have error surfaces with broader valleys, while the error surfaces of difficult datasets have only narrow ravines.

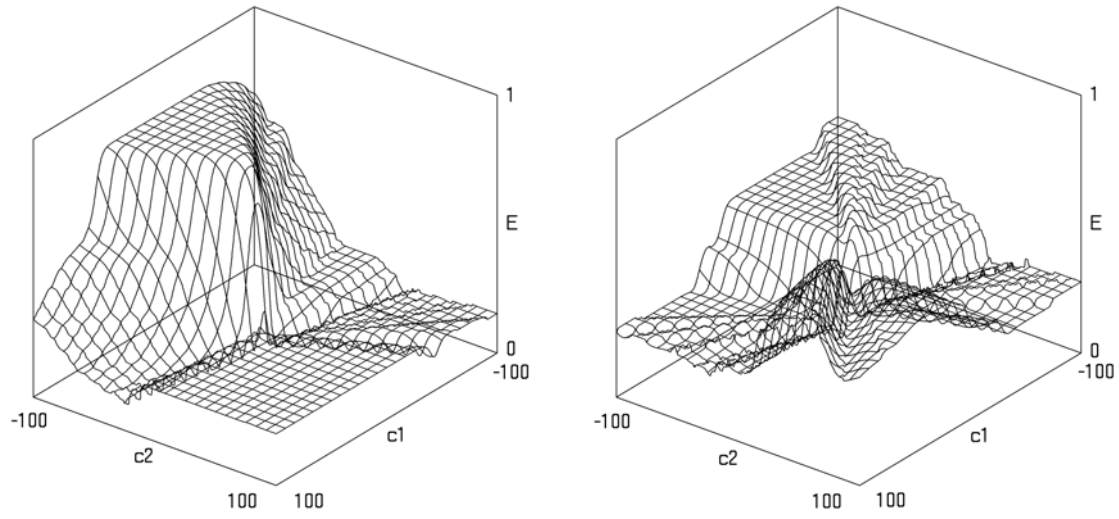

Fig. 1.16. Left: ES of entire Appendicitis dataset (7-4-2). Right: ES of Appendicitis dataset (7-4-2) with only 42 vectors – all 21 vectors of class 1 and randomly chosen 21 vectors of class 2.

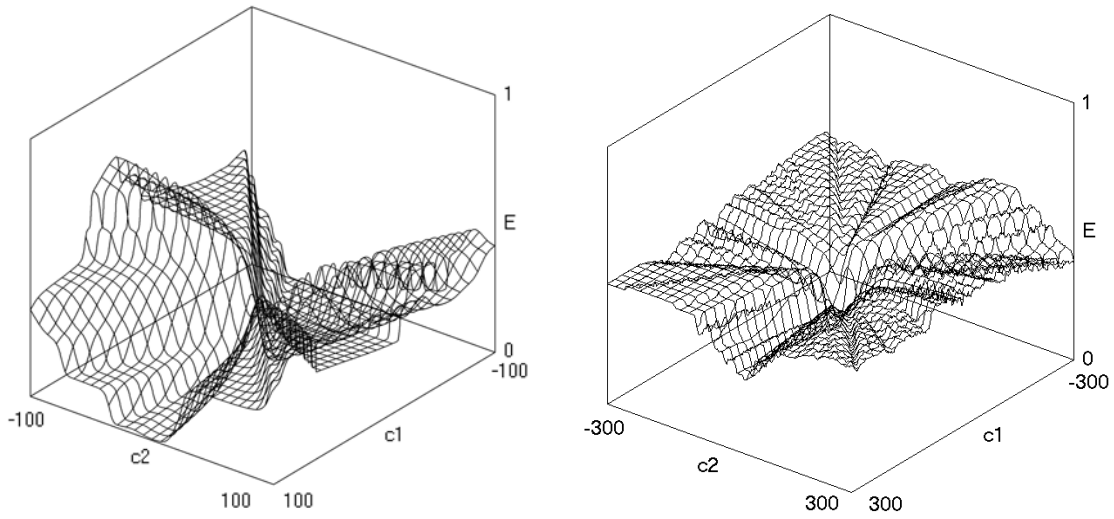

Fig. 1.17. Left: ES of Xor (2-2-2). Right: ES of 6-bit parity (6-8-2).

1.2.6. Transfer Function Influence on Error Surface

1.2.6.1. Monotone Transfer Functions

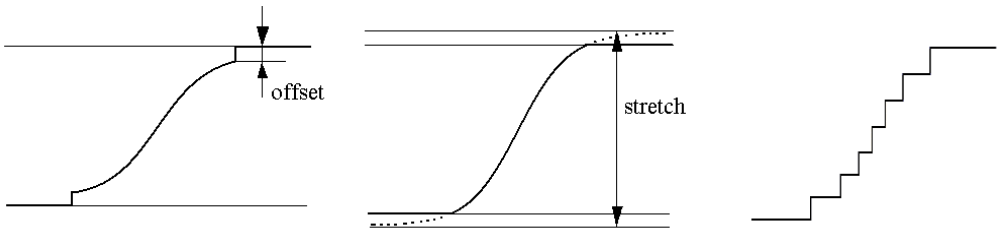

Fig. 1.18. Transfer functions: a) sigmoid with offset, b) stretched sigmoid, c) staircase function.

This chapter contains examples of error surfaces with various transfer functions, such as a sigmoid with offset, a staircase function and a stretched sigmoid. The purpose of introducing the functions is to prevent the weights from an infinite growth and in the case of a staircase function also to simplify the calculations.

Discontinuities are visible in the plot of ES obtained with a staircase function and with a sigmoid with offset. Both functions give a similar ES (Fig.1.19-right) with the distinguished feature of sharp edges. The differences are visible in a smaller scale; the sigmoid with offset gives smooth surfaces with curbs (Fig.1.20-left), while the staircase function produces quite irregular surfaces (Fig.1.20-right). Both the offset increase and the decrease of the number of stairs make the training more difficult and produce sharp edges on the ES. Moreover, these transfer functions are not continuously differentiable and impose problems to analytical gradient-based methods. The stretched sigmoid does not cause any

sharpness on the error surface and in this way it differs from the two previous transfer functions. With a small stretch (1.01÷1.1) it seems to be an optimal solution. But with a bigger stretch the function becomes similar to a step function and has a limited usefulness for complex datasets – the error surfaces are becoming simple with big flat areas (Fig.1.19-left).

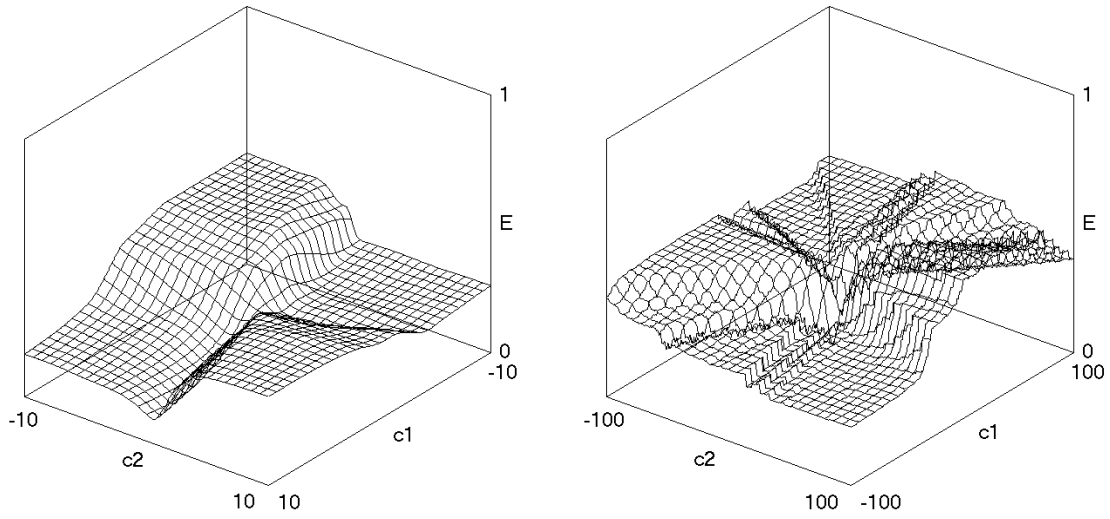

Fig. 1.19. Left: ES of Ionosphere (34-4-2) with stretched sigmoid (stretch=1.3). Right: ES of Iris (4-4-3) with staircase transfer function (5 stairs).

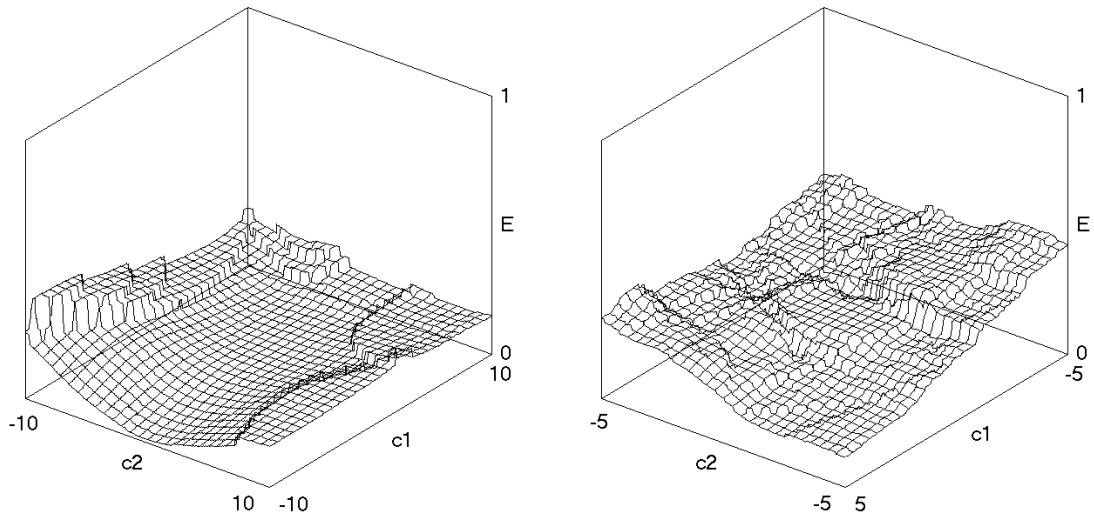

Fig. 1.20. Left: ES of Iris (4-4-3) with sigmoid with offset=0.2 visible with large zoom. Right: ES of Iris (4-4-3) with staircase function (11 stairs) visible in a big zoom.

1.2.6.2. Non-monotone Transfer Functions

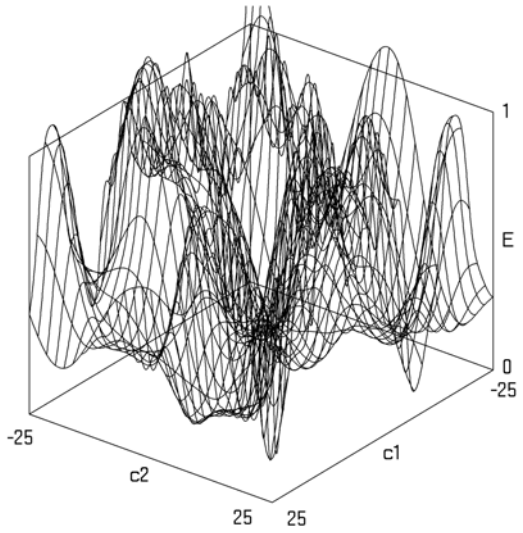

Fig. 1.21. ES of Xor (2-2-2) with sinusoidal transfer function S=0.3+0.9·sin(0.3·x).

Non-monotone transfer functions produce lots of local minima. Fig.1.21 shows ES of Xor (2-2-2) with a sinusoidal transfer function. The training of the network was successful because during the training all weight remained in the monotone interval of the sinusoid ($-\pi/2$; $\pi/2$). ES visible in this figure has nothing in common with ES of MLPs with monotone transfer functions, such as widely used logistic sigmoid and hyperbolic tangent, where local minima are very rare for real-world datasets, although they may exist as an effect of superpositions of two or more sigmoids. Mainly an ill-conditioning, large flat areas and choosing a wrong ES ravine cause many difficulties for training algorithms.

1.2.7. Local Minima

The most well-known difficulty that arises in general optimization problems is the issue of local minima. Mathematical programming and optimization research was originally concerned with univariate problems, or with solving systems of equations involving only a few variables. In the one-dimensional case, the concept of local minima follows closely from the issue of convexity. The conceptual picture is that if there are no local minima, then the optimization problem is trivial, and the cost function resembles a parabolic bowl or a single valley. This picture has persisted in MLP research, perhaps mainly because it was used to explain the failure of backpropagation to learn, and because the large amount of techniques from optimization being applied to the development of training algorithms [Gallagher 2000].

Rumelhart stated that the occasional failure of MLPs to learn simple problems including Xor was caused by local minima [Rumelhart 1986b]. This together with the experience from the low-dimensional optimization problems led to a widespread perception that local minima are the greatest obstacle in successful MLP learning (if the training was

successful, then the algorithm found a global minimum, whereas if the training did not progress satisfactorily then the algorithm was stuck in a local minimum). A good example of the widespread improper conceptual picture can be found in [Wilson 2003], where a picture very similar to Fig.2.21 is placed followed by a comment "there may be many thousands of weights, making the error surface difficult to visualize".

Some authors claimed that the ES of Xor 2-2-1 and Xor 2-1-1 (with cross-over connections) contain local minima [Blum 1989] [Lisboa 1991] [Gori 1992] [Horikawa 1993] by which backpropagation can become trapped.

However, a more detailed analysis of the problem revealed that the error surface of both Xor 2-1-1 [Sprinkhuizen 1996] and Xor 2-2-1 [Hamey 1995] [Hamey 1998] networks have no local minima. All stationary points in the 2-1-1 Xor problem are saddle points and there exist finite trajectories, which allow escape, without increasing the error, from all finite stationary points. Thus the points are not local minima. It was also shown [Sprinkhuizen 1998] that all stationary points with finite weights are saddle points with positive error or zero error and not local minima.

Overall, the analysis of the Xor error surface indicates that local minima are not the cause of poor training performance for algorithms such as backpropagation. Other features, such as saddle points and plateaus, seem more likely explanations of training difficulties. Coetze [Coetze 1997] indicates that empirical MLP error surfaces have an extreme ratio of saddle points to local minima.

It is known that MLP error surfaces are often ill-conditioned [Le Cun 1991], [Saarinen 1993], with the Hessian eigenvalues differing by orders of magnitude. This fact means that there are often directions on the error surface in which the gradient varies quickly (cliffs or steep ravines) and others, where the gradient variation is quite slow (plateaus or flat regions) [Hecht 1990] [Lehr 1996]. For an algorithm such as backpropagation with a fixed step size, this feature leads to periods of very slow progress, sudden drops and oscillations in the error values.

There are several factors that contribute to the ill-conditioning in MLP error surfaces. The properties of transfer functions are reflected in the properties of the error surface, as it was seen in the ES projections obtained with various transfer functions. The sigmoids and still more their superpositions cause the ill-conditioning. Attempting to make sure the sigmoids in the network operate effectively in their useful regions is one way to reduce the effects of ill-conditioning [Le Cun 1998]. Very small training sets may also contribute to ill-conditioning [McKeown 1997].

The local minima were never visible in the ES projections, while the ill-conditioning effect was frequently. Though it was shown that local minima can exist [Sontag 1989], they are important mostly from the theoretical point of view, while ill-conditioning has much more direct and practically important effect on the training algorithms performance.

1.2.8. Error Function Influence on Error Surface

Using MSE error function with desired output signals 0.1 and 0.9 (or 0.2 and 0.8) produces very similar ES as with desired outputs 0 and 1 but a global minimum tends to lie close to the ES center in a shallow valley (not shown here).

1.2.8.1. Different Exponents in Error Function

An error surface depends also on the power exponent of the error function. Typically MSE functions (exponent=2) are used but for exponents ranging from 0.5 to 8.0 error surfaces look very similar to those obtained with MSE.

Two plots of error surfaces obtained with the exponent = 0.1 and 32 are shown here. High error exponents successfully reduce the weight growth and can be used as a weight regularization method. The learning trajectory remains near the ES center. For Iris (4-4-3) the length of the weight vector never exceeded 25, no matter how long the training was and the network was always successfully trained. Low exponents produce ES with relatively high plateaus and the slopes the ES fall down very slowly. With the exponent = 0.1 it is usually enough to reduce the distance error by 20% to achieve the same classification accuracy on a training set, as would require reducing MSE by 90%. However, the network training with such low exponents of the error function may be difficult.

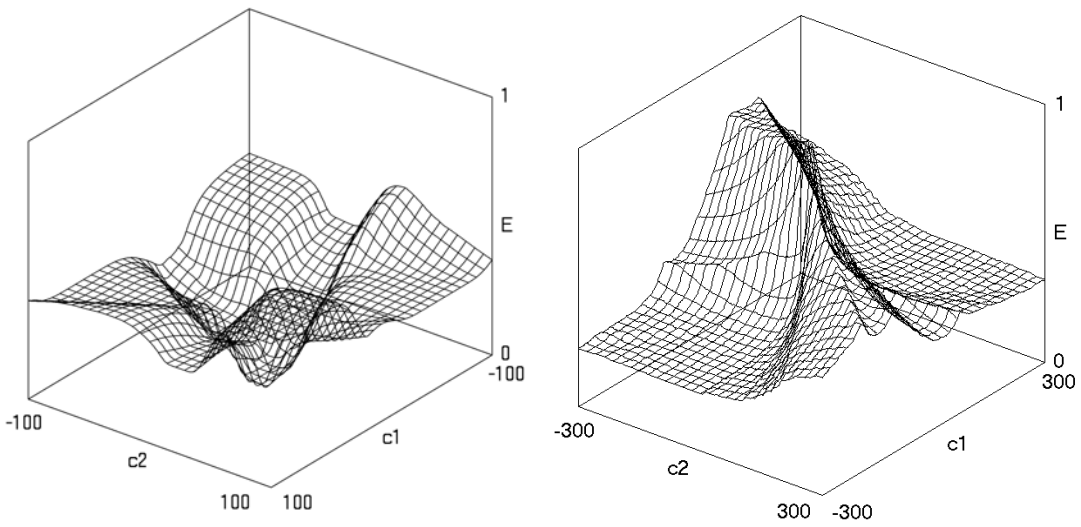

Fig. 1.22. Left: ES of Iris (4-4-3) with power function exponent=32. Right: ES of Iris (4-4-3) with power function exponent=0.1.

1.2.8.2. Weight Regularization

The regularization term is added to the error function to prevent the weights from excessive growth in order to provide better generalization (chapter 2.6.3). In the simplest weight decay model the penalty term for big weight values is added to the error function as the sum of the weight squares. The error function is:

$$E = \sum_v \sum_c f(d_{v,c} - s_{v,c}) + c \sum_i w_i^2 \tag{1.19}$$

The error surface then lifts up, less near the center and more further from the center, thus we can see a superposition of the original ES with the paraboloid caused by the regularization term. The effect is stronger for bigger *c* values. A plot for the Breast dataset with *c*=0.03 is presented in Fig.1.23.

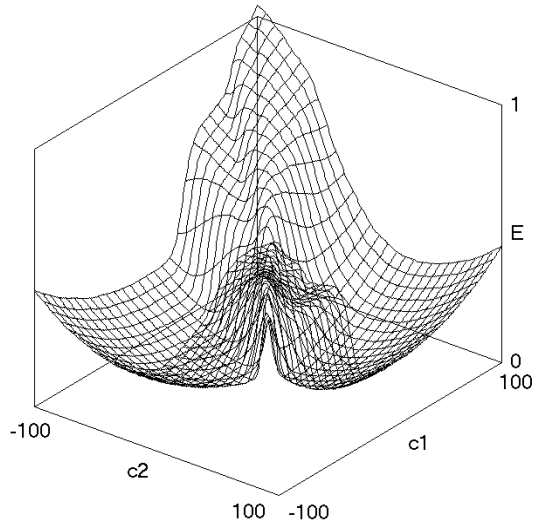

Fig. 1.23. ES of Breast (10-4-2) with weight regularization, c=0.03.

1.2.8.3. Cross-Entropy Error Function

Solla [Solla 1988] showed that for a cross-entropy error measure, the error surface is steeper in the region of a minimum, in comparison to MSE function. Thus, using the cross-entropy error function can improve the network convergency close to the minimum.

The cross-entropy error function is given by the following formula:

$$E = -\sum_v \sum_c \left(T_{vc} \ln S_{vc} + (1 - T_{vc}) \ln(1 - S_{vc}) \right) \tag{1.20}$$

where *v* is the vector number, *c* is the output neuron number, corresponding to the class number, *T* is the desired output neuron signal and *S* is the actual output neuron signal The error is summed over all vectors *v* and all output neurons *c*. But since for (*S=0 ,T=1*) and (*S=1,T=0*) the function takes infinite values the following modification is used:

$$E = -\sum_{v}\sum_{c}\left(T_{vc}(\ln(S_{vc}+d)-\ln(1+d))+(1-T_{vc})(\ln(1-S_{vc}+d)-\ln(1+d)\right) \tag{1.21}$$

where d is a small (about 10^{-10}) positive number.

Comparing to MSE or other power error functions, cross-entropy error functions give similar or more complex ES. Fragments of the ES are higher then 1, due to the fact that the error is not bounded by $N_v N_c$ as in the case of power error functions.

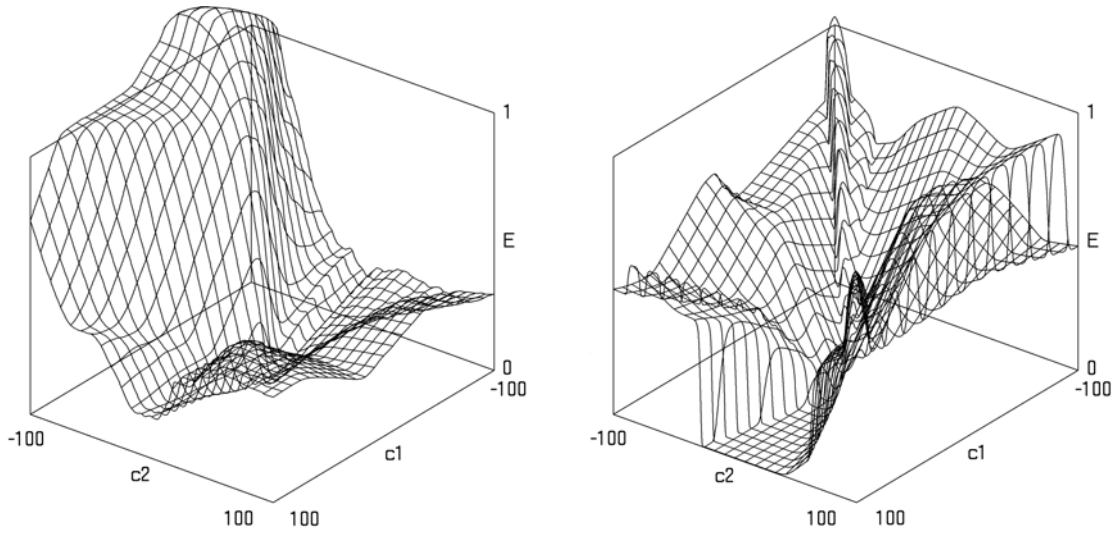

Fig. 1.24. Left: ES of Appendicitis (7-4-2) with cross-entropy error function. Right: ES of Xor (2-2-2) with cross-entropy error function.

1.2.9. Weight Changes on Error Surface

Using the principal components, from the equation (1.12) we can calculate the weight values at any point of the projected error surface:

$$\boldsymbol{W}(c_1,c_2)=\begin{bmatrix} w_1(c_1,c_2) \\ \vdots \\ w_n(c_1,c_2) \end{bmatrix} = \begin{bmatrix} w_{01} \\ \vdots \\ w_{on} \end{bmatrix} + c_1 \begin{bmatrix} \lambda_{11} \\ \vdots \\ \lambda_{1n} \end{bmatrix} + c_2 \begin{bmatrix} \lambda_{21} \\ \vdots \\ \lambda_{2n} \end{bmatrix} \tag{1.22}$$

On average the weight values in the areas of lower error are more symmetric with respect to zero and the disproportions between the values of different neuron weights are smaller. However, the tendencies are not very strong.

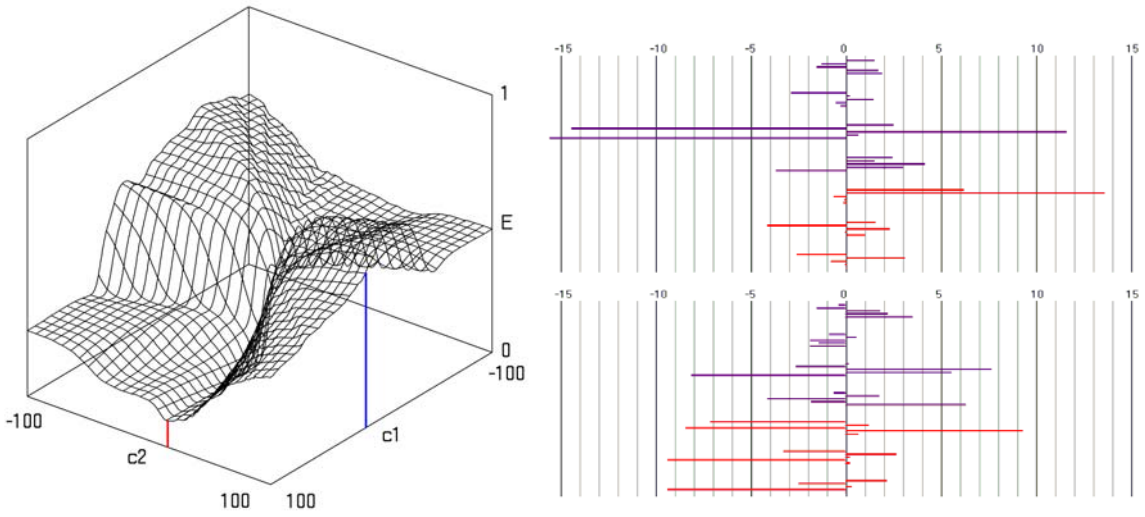

Fig. 1.26. Left: ES of Iris (4-4-3). Right top: weight values in the point with low error (red line intersecting the ES). Right bottom: weight values in the point with high error (blue line intersecting the ES). In violet: hidden neuron weights, in red: output neuron weights.

1.2.10. Reducing the Number of Effective Parameters

PCA is a well-known technique, widely used for the preprocessing of training data to reduce the number of network inputs. PCA was also proposed for weight pruning [Levin 1994]. In this section the possibility to use of PCA to reduce the number of effective training parameters is discussed. After training the network for some epochs, PCA is performed on the weight covariance matrix. Then searching for the error minimum takes place in the reduced space of PCA-determined directions.

1.2.10.1. Directions in the Weight Space

The analysis of directions in a weight space reveals interesting properties of ES that can be used to design or improve some neural training algorithms [Kordos 2004b]. Some trends and tendencies are common for many datasets and network structures with differences only in details.

The cosine of the angle between two vectors $\boldsymbol{A}=[a_1, a_2, \ldots a_n]$ and $\boldsymbol{B}=[b_1, b_2, \ldots b_n]^T$ can be calculated as:

$$\cos(\boldsymbol{A},\boldsymbol{B}) = \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{|\boldsymbol{A}||\boldsymbol{B}|} = \frac{a_1 b_1 + \ldots + a_n b_n}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{1.23}$$

The lines for *cos W, ||W||, E, cos(W,PC)* shown in Fig.1.27-left look very similar for various training methods (the sample training was performed on the Iris dataset using backpropagation with variable learning rates). It can be seen that the error *E* decreases

proportionally to the changes of weight vector direction *cosW*. At the final stage of the training, the direction remains almost constant and the error is decreasing very slowly, although the weights are still growing. The trajectory is then already in the flat part of the ES. In some cases, such as weight regularization, the weights do not grow to infinity, but only to limited values, nevertheless the error decreases as long as the weight vector changes its direction.

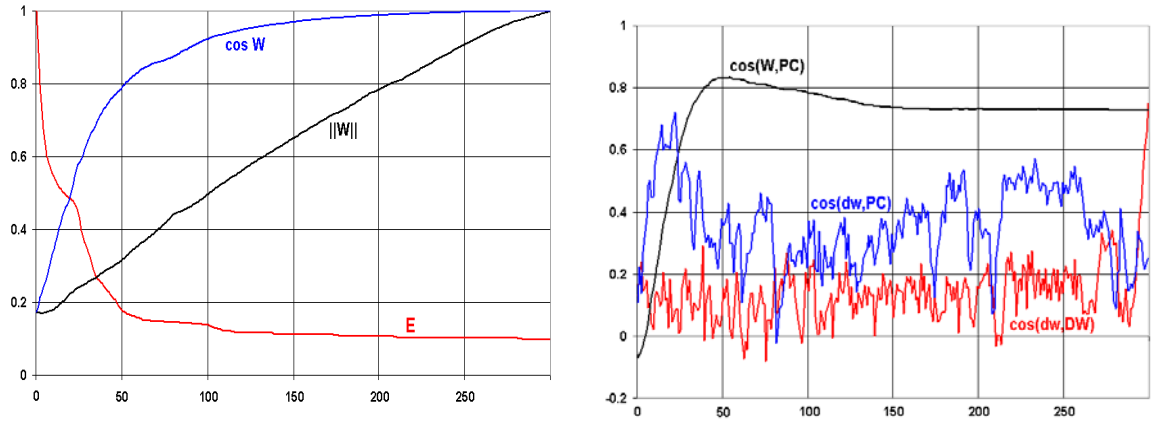

Fig. 1.27. Left: Change of parameters during network training. Vertical axis: normalized ||**W**||, normalized (rescaled to 1) MSE and *cos**W**=cos(**W**[epoch]-**W**[last epoch])*. Horizontal axis: epoch number. Right: Change of parameters during network training. Vertical axis: *cos(W,PC)*=cosine between the weight vector ***W*** and the first PCA direction, *cos(dw,DW)=cos(**W**[epoch]-**W**[epoch-1],**W**[last epoch])*, *cos(dw,PC)=cos(**W**[epoch]-**W**[epoch-1],**PC**)*. Horizontal axis: epoch number.

The line *cos(W,PC)* in Fig.1.27-right shows the cosine of the angle between the first PCA direction and a line connecting the starting point with the actual trajectory point. Only the weights from the first 100 epochs were included in the weight matrix for PCA calculation. The cosine takes the greatest value about the 50th epoch. Afterwards PCA and ***W*** directions diverge. The divergence is sometimes even stronger than in Fig.1.27-right. For that reason a big jump only seldom can be made along PCA directions while training the network. PCA directions are very good for ES and even better for trajectory visualization, where a little difference in angles does not matter. However using PCA for learning trajectory extrapolation, thus making a jump several epochs ahead, is not an easy task, since the proper direction of the jump must be determined very precisely.

The two other lines (red and blue) in Fig.1.27-right show the cosine of angles between the temporary direction of the trajectory (a vector connecting the last and the actual trajectory point) and the first PCA direction *cos(dw,PC)* and between the temporary direction of the trajectory and the direction determined by the starting and the last trajectory point *cos(dw,DW)*. These two characteristics differ strongly depending on a training algorithm. The values of some other angles are shown in Table 1.2.

Table 1.2. Cosine between particular directions in the MLP weight space for the same training as in Fig.1.27.

	PCA_c1 (λ_1)	PCA_c2 (λ_2)	||1||	traject
ICA_c1	-0.99585	0.08037	0.05832	-0.88185
ICA_c2	-0.96653	-0.15675	0.1038	-0.75964
||1||	-0.06087	-0.02601	1	-0.04991
traject	0.86626	-0.25371	-0.04991	1
PCA_c1 (λ_1)	1	0	-0.06087	0.86626
PCA_c2 (λ_2)	0	1	-0.02601	-0.25371

traject – direction of a line connecting the first and last trajectory point
||1|| – direction of the diagonal vector [1,1,1,...,1]

1.2.10.2. PCA-based Parameters Reduction. A Case Study

1. Starting from the random weights (error=326) the network (10-4-2) is trained on the Wisconsin Breast Cancer dataset for some training cycles using numerical gradient (chapter 2.2). The training is stopped with the error=240.
2. PCA directions are determined.
3. A minimum in the PCA directions is found (also using numerical gradient) with the error=43 and a jump is made to that point (blue arrow in Fig 1.15-left).
4. No further error decrease in PCA-directions is possible. The network is trained again with a standard numerical gradient for 5 training cycles.
5. Again, PCA directions are determined on the weight matrix from the last 5 training cycles.
6. PCA provides the eigenvectors that determine only the directions, the constant values must be added to the weights. The values do not have to be the mean values that were subtracted from the weights while calculating the covariance matrix (equation 1.7). We would rather like them to be the values of the last trajectory point, since this ensures that the training in the reduced weight space can start from the last point of the training in the full weight space. However three possibilities of choosing the point (called "fixing point") are considered:
   a) The zero point in the weight space. However this causes that the projection of the ES lifts up. The lowest point on it has now the error=244 (Fig.1.28-left)
   b) The point of the mean weight values. The obtained ES looks like an intermediate stage between Figs. 2.28-left and 2.28-right. Moreover the point of mean weight values is usually not contained in the learning trajectory and much higher error can correspond to that point.
   c) The last training point (Fig.1.28-right). This is the only reasonable choice. When the point is chosen as a fixing point, the projection of ES does not lift up, but because PCA directions are determined on the weight matrix from only a small part of the training, we get some local PCA directions. In the local PCA directions the minimum is situated very close to the last training point.
7. Thus, the big jump several training cycles ahead could be made only once.

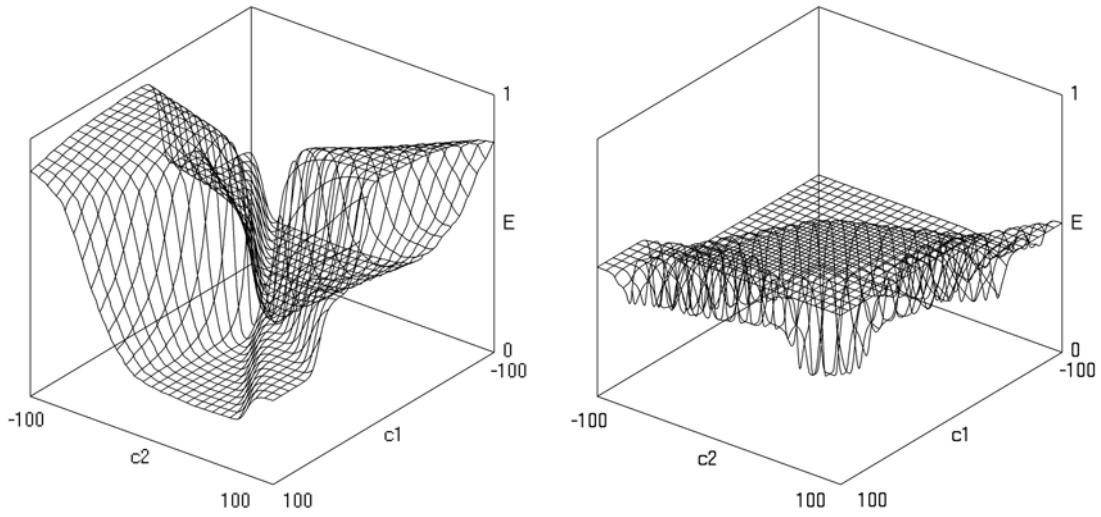

Fig. 1.28. ES of Breast (10-4-2) determined basing on 5 training cycles after the jump. Left: fixing point at the zero point in the weight space. Right: fixing point at the last trajectory point.

Breast dataset was chosen for the case study intentionally because the dataset is very easy to train (what is clear, since its ES is very simple). For most datasets such big jumps in PCA directions (from error=240 to 43) are impossible. However, using a PCA-based ES projection on which the last training point is situated, it is usually possible to find a point with lower error (Fig.1.28-right). The reason for which the method is in most cases impractical is the computational cost of calculating PCA every some epochs in order to make only a small step in the reduced space.

1.2.11. Sections of MLP Error Surface

MLP error surface changes slower in the parts that are located further from its center. These parts are reached by the learning trajectory at the final stage of the training. However mostly output layer weights contribute to the slower changes. At the beginning of the training (close to the ES center) usually the error function derivatives in output layer weight directions are bigger, although the distances from the actual point to the error minimum are shorter. That is quite opposite to BP assumptions. (There are also versions of BP that use different learning rates in different layers.) Therefore RPROP, which takes into account only the sign of a derivative, performs not worse than BP. At the final training stage the landscape changes, but mainly in output layer weight directions. The differences between error surface sections in hidden weight directions at the beginning and at the final stage of the training are not so significant. In any case, gradient direction is not the optimal next step direction.

Frequently some features are irrelevant for the classification task. Error surface sections in the directions of the weights that connect the irrelevant inputs with hidden neurons are almost flat. They may only slightly change due to random noise contained in the features.

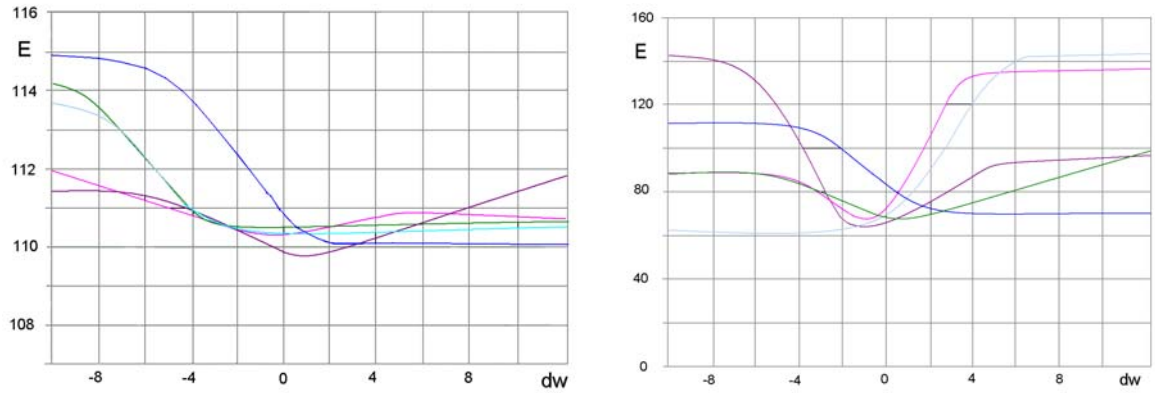

Fig. 1.29. Left: ES sections in hidden weight directions in the first NG (numerical gradient) training cycle. Right: ES sections in output weight directions in the first NG training cycle.

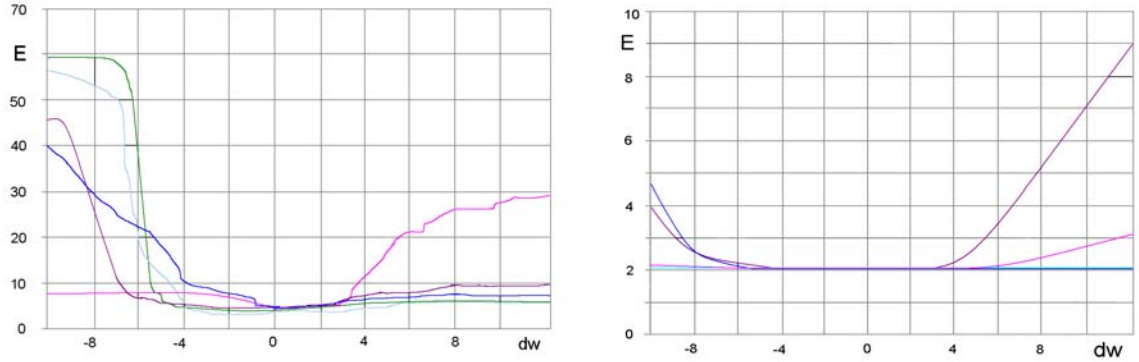

Fig. 1.30. ES sections in hidden weight directions in the 15th NG training cycle. Right: ES sections in output weight directions in the 15th NG training cycle.

The observations can be practically used for training algorithm optimization and they have been implemented into numerical gradient algorithm thus achieving significant reduction of training times (chapter 2.3.5).

1.2.12. Conclusions

Although it is impossible to see n-dimensional spaces in 3 dimensions without any distortions, the first and second PCA component coordinate system gives quite a good insight into many important ES properties, which are listed below:

- ES of MLP networks has a starfish structure.
- ES depends on network architecture and training data as well as on transfer and error functions.
- Local minima in craters are very rare in standard MLP networks with monotone transfer functions trained on real-world datasets.
- With MSE error function and sigmoidal transfer functions global minima are in infinity in the ravines reaching the lowest error values.

- With MSE error function and sigmoidal transfer functions local minima are also in infinity but in the ravines reaching higher error values.
- Ill-conditioning, large flat areas, or choosing a wrong ES ravine due to a poor weight initialization may cause many difficulties for training algorithms.

The training method used to generate data for PCA does not significantly influence the ES projection shape. The learning trajectories of many algorithms create an arc lying on the bottom of one of ES ravines, though the arc may be smoother or rougher. The view of ES projection depends on the weights after each epoch. If the training is not successful than the learning trajectory does not traverse enough space and both the trajectory and the ES projections are too flat and too highly situated.

The shape of ES has the greatest diversity close to its center. Far from the center, the surface changes slowly and flat horizontal areas occupy much place. If the random initial weight range is too broad then there is a great chance that the starting point lies somewhere on the flat area, and as a result the network cannot be trained with any gradient-based or local search methods. On contrary, if all initial weights are zero, the network can be successfully trained with appropriate methods, such as VSS, because gradients are big in this point. It cannot in this case be trained with BP or NG, but this is due to the limitations of the training methods and not of the ES properties around the zero point.

In some cases the network training can be accelerated by determining PCA components in the weight space after some initial training and then jumping to a minimum found in PCA coordinates or by extrapolating the learning trajectory in PCA directions. However, a universal solution has not been found so far. Non-linear techniques, such as principal curves, principal surfaces or kernel PCA, can also be used to display the surfaces and to attempt the reduction of training times. This may be one of the future research subjects aimed at a better understanding of neural networks and improving network architectures and training methods.

1.3. Visualization and Properties of MLP Learning Trajectories

1.3.1. Error Surface and Learning Trajectory

In the 3-dimensional plots the learning trajectory usually intersects the error surface in only one point that will be called a „fixing point" and that corresponds to $\boldsymbol{W_0}$ in the equation (1.12). One of the learning trajectory points can be arbitrary selected as the fixing point, while placing in the same plot the error surface and the learning trajectory projections. Using the equations (1.10) and (1.11) does not always work well because the point given by (1.11) is usually not traversed by the learning trajectory.

The convention in this thesis is that the zero point in the weight space is always in the middle of the base of the cube (c1=0 and c2=0). In all figures presented so far ES projections were fixed to the zero point in the weight space (except Fig. 1.28-right). Although such ES projections are very similar to ES projections fixed to a given point of the learning trajectories, they do not adhere to the trajectories well.

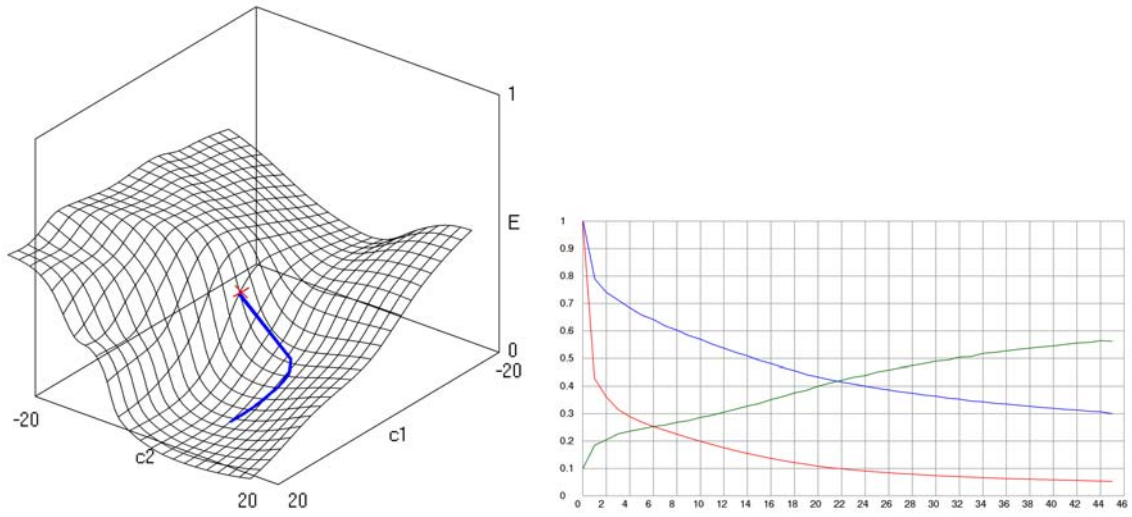

Fig. 1.31. Left: ES and learning trajectory of Ionosphere (34-4-2). Right: vertical axis: training error in the point (c1,c2) (red) and corresponding error value on the ES projection (blue), horizontal axis: training cycle. Fixing point (red cross) at the starting trajectory point.

The learning trajectory obviously does not lie on the two-dimensional ES projection, but somewhere in the multidimensional weight space and therefore it cannot ideally adhere to the ES projection. The first and second PCA components comprise typically 95-97% of the total variance in the weight ("horizontal") directions. Nevertheless, the information about the error value is not included in PCA calculations. Therefore, it may happen that although the "horizontal" distance between the original multidimensional trajectory and its projection into the first and second PCA direction is within 5% accuracy, the network error in the two points may differ much more. The effect is caused by a high nonlinearity of the error surface.

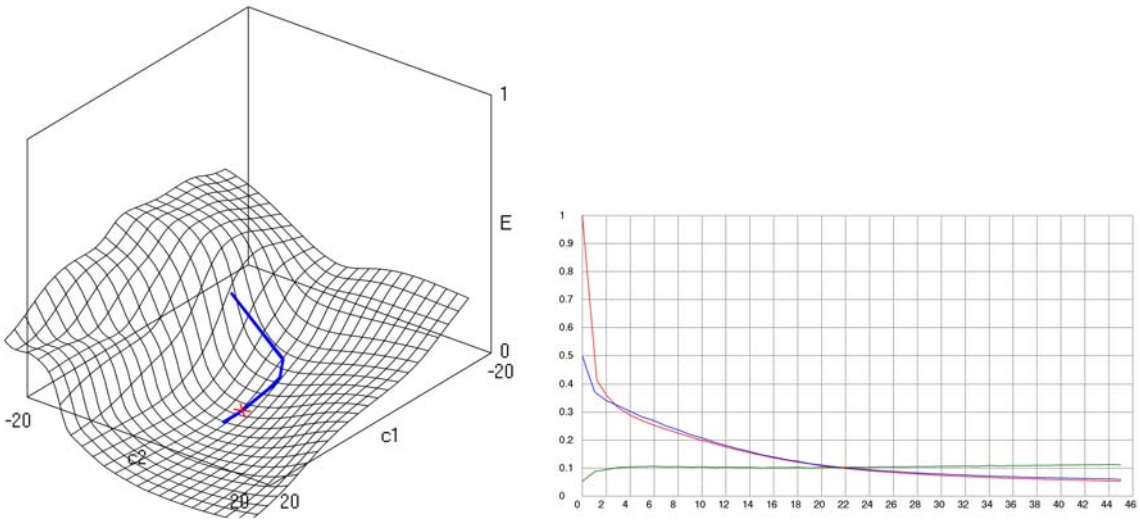

Fig. 1.32. Left: ES and learning trajectory of Ionosphere (34-4-2). Right: vertical axis: training error in the point (c1,c2) (red) and corresponding error value on the ES projection (blue), horizontal axis: training cycle. Fixing point (red cross) at the 20th training cycle.

However, such rapid changes in the error surface are relatively rare. In most of network trainings the learning trajectory adheres to the ES projection relatively well along quite a significant fragment around the fixing point (Fig. 1.32). For small networks and simple datasets the good adherence can be obtained for the entire trajectory (Fig. 1.33). The vertical coordinate of a given trajectory point corresponds to the factual network error during the training, whereas the vertical coordinate of the ES projection point (which has the same horizontal coordinates as the given trajectory point) shows the error calculated using only the first and second PCA component.

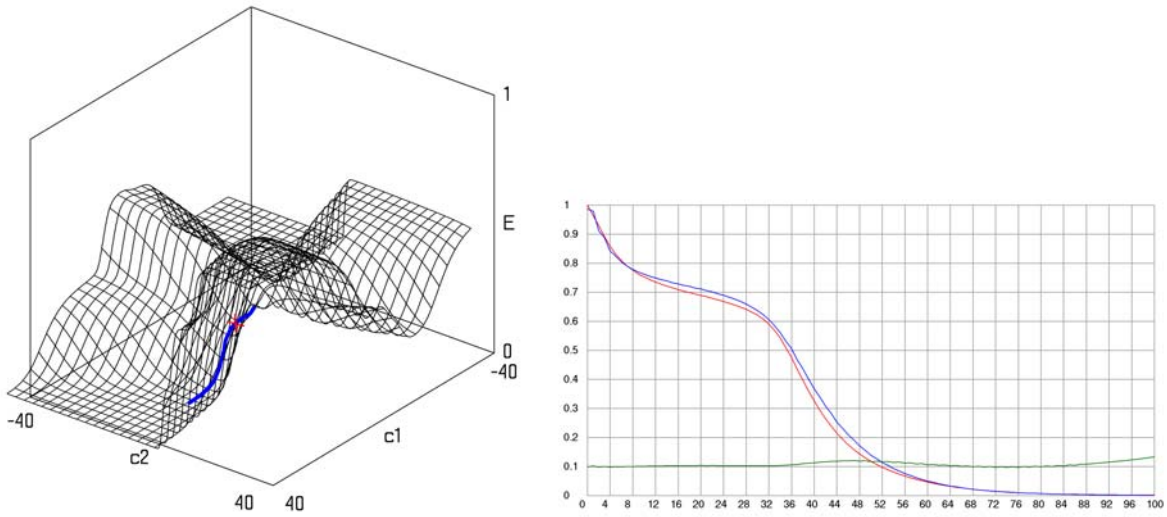

Fig. 1.33. Left: error surface and learning trajectory of Xor (2-2-2). Right: vertical axis: training error in the point (c1,c2) (red) and corresponding error value on the ES projection (blue), horizontal axis: training cycle. Fixing point at the 7th training cycle.

Even if the trajectory does not adhere to the ES projection well, it at least shows us which ravine was chosen by a training algorithm. The trajectories in n-dimensional weight space frequently create arcs. The mean direction of the arc usually corresponds to the direction of the ES ravine in PCA projections. The beginning of a trajectory (the training cycles before the fixing point) lies often over the ES projection and its end (the training cycles after the fixing point) under it. Thus, the ES projections are often flatter then original ES on which the trajectories lie.

1.3.2. Learning Trajectory Extrapolation

PCA projections are most reliable and the original proportions are best preserved if PCA directions are determined using the weights from all training cycles (Fig. 1.33). If PCA is calculated using only weights from a fragment of the training and the entire learning trajectory is projected into so obtained PCA directions than the fragment of trajectory included in PCA calculations not only tends to be magnified but also has a higher ratio of its size in c2 to its size in c1 direction, what is visible as ”bigger teeth” (Fig. 1.36-1.37). Moreover, quite irrelevant results are obtained outside that fragment (Fig. 1.34). That is clear because the remaining data is projected using not its own PCA directions.

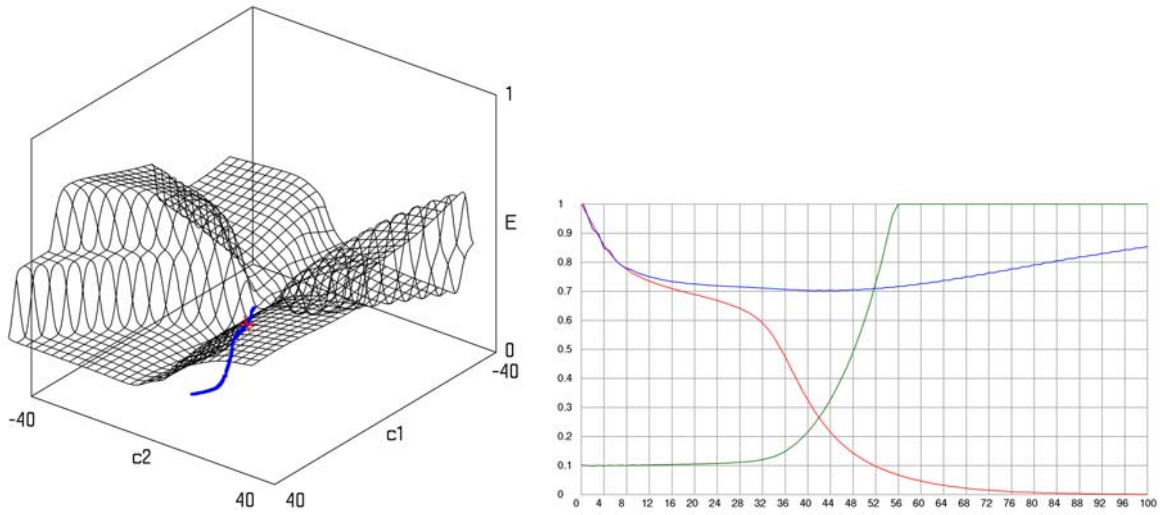

Fig. 1.34. Left: error surface and learning trajectory of xor (2-2-2). Right: vertical axis: training error in the point (c1,c2) (red) and corresponding error value on the ES projection (blue), horizontal axis: training cycle. PCA was calculated on weights from the training cycles 0...10. Fixing point at the 7th training cycle.

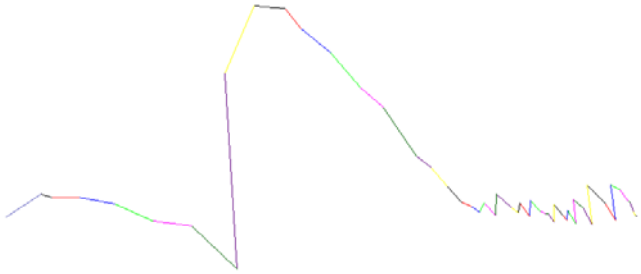

FIG. 1.35. Projection of the Iris (4-4-3) learning trajectory trained with NG with an oversized step in the first and second PCA direction. PCA was calculated on weights from the entire training (cycles 0...50). The color changes every training cycle.

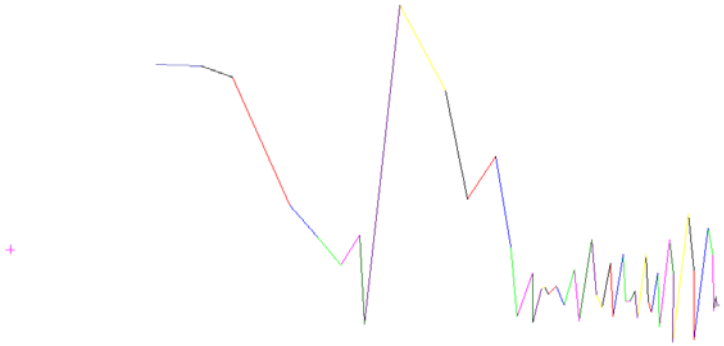

Fig. 1.36. Projection of the Iris (4-4-3) learning trajectory trained with NG with an oversized step in the first and second PCA direction. PCA was calculated on weights from the training cycles 20...55. The color changes every training cycle.

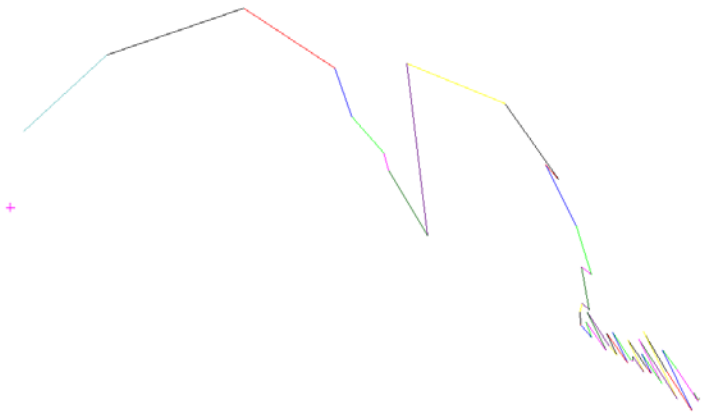

Fig. 1.37. Projection of the Iris (4-4-3) learning trajectory trained with NG with an oversized step in the first and second PCA direction. PCA was calculated on weights from the training cycles 0...5. The color changes every training cycle.

1.3.3. Learning Trajectories of Various Training Algorithms

The shape of a learning trajectory depends on all parameters that influence the shape of ES and additionally on the training algorithm and its parameters. For example, BP with a small learning rate produces very smooth trajectories. Increasing learning rate gives more irregular trajectories [Gallagher 2003]. Fragments of the BP trajectories may go as well downwards as upwards, while trajectories obtained with some other algorithms (NG, VSS) go only downwards.

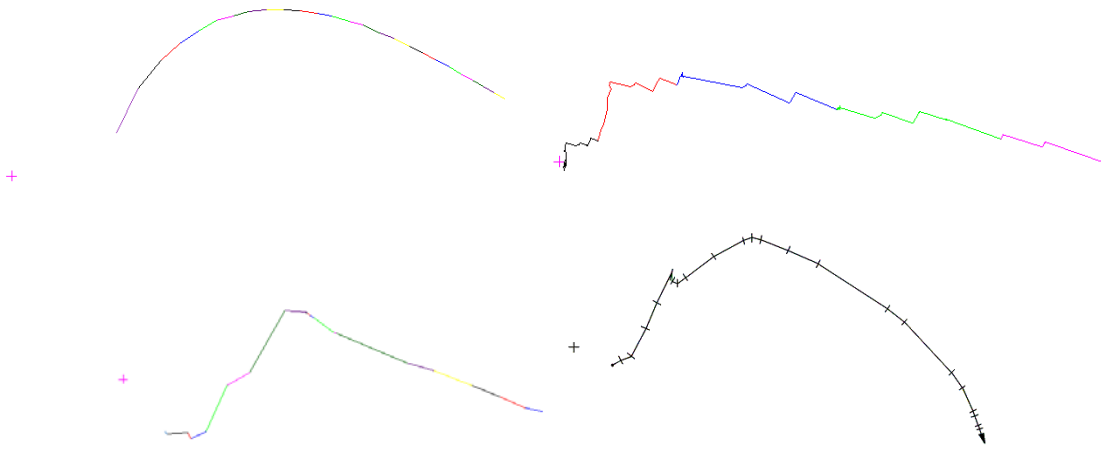

Fig. 1.38. Projection of the Iris (4-4-3) learning trajectory trained with various algorithms (1-NG, 2-VSS, 3-LM, 4-SCG, in the first and second PCA direction. The cross shows the zero point in the weight space. The color changes every training cycle.

Only the trajectories of batch versions of learning algorithms are shown here. The ES is strictly associated with a given set of vectors. In any training, which does not calculate the error on the entire set (e.g. online backpropagation), a different ES corresponds to a different subset of training vectors. It would be impractical to show such trajectories for two reasons: first it is unclear in what coordinate system they should be shown and second the trajectory

fragments in online trainings are very small and it would be difficult to see them in the entire trajectory scale.

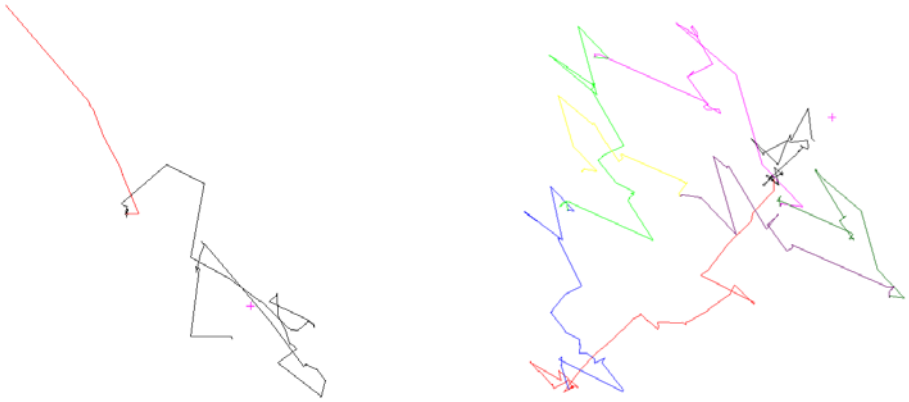

Fig. 1.39. Left: Projection of the very beginning of the Iris (4-4-3) learning trajectory trained with VSS in the first and second PCA direction. Right: Projection of the Iris (4-4-3) learning trajectory trained with VSS in the third and forth PCA direction.

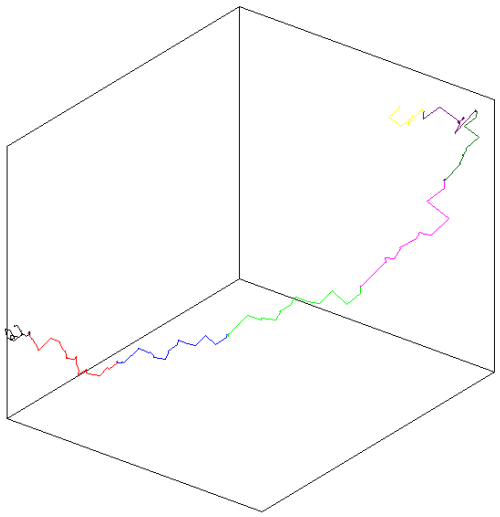

Fig. 1.40. Projection of the Thyroid (21-4-3) learning trajectory trained with VSS in the first (c1), second (c2) and third (c3) PCA direction. Horizontal axes: c1 and c2, vertical axis: c3.

In spite the fact that learning trajectories look differently for different training algorithms, the first and second PCA directions usually capture together about 95-97% of their total variance. Thus, the PCA projections of learning trajectories reflect the properties of the original trajectories quite well. The similarity between all trajectories presented in Figs. 1.38 is obvious; they create similar arcs following the shape of the Iris error surface ravine. The differences between them will be discussed in chapter 2.4.4. The easier the training of

the dataset is the simpler and more regular is the learning trajectory. The Iris dataset is relatively easy for training and its learning trajectories create regular arcs. Higher PCA components have significant values only at the beginning of the training, what is clear because at that stage training algorithms chose the proper direction. As the training approaches the final stage, the direction changes are usually small.

1.4. Weight Changes during MLP Training

This chapter contains only a short review of the properties of weight changes that are common for local MLP training algorithms (analytical gradient-based and search-based). The changes of weight values during network training depend on the shape of the error surface as well as on the training algorithm. This analysis proved to be a useful factor when designing VSS algorithm.

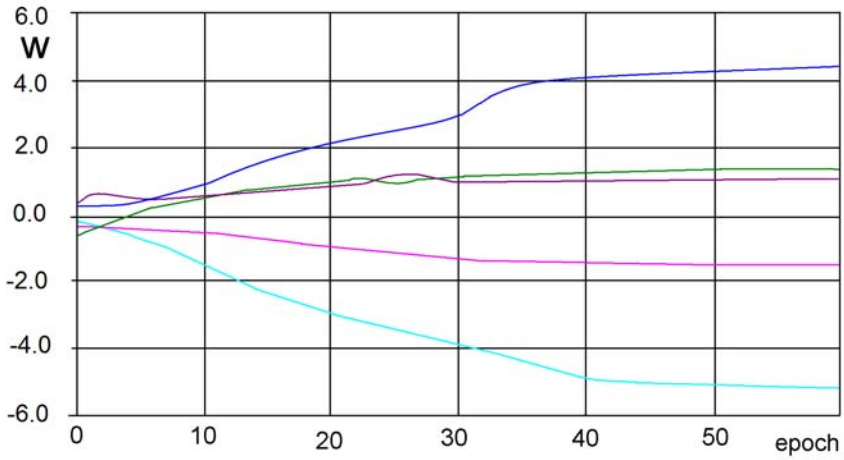

Fig. 1.41. Typical changes of weight values during MLP training with local methods.

In general, three properties can be noticed. First, on average the change of a given weight value in the actual epoch is similar to its change in the previous epoch. Second, the final values of different weights can vary ranks of order. Third, after several training epochs some weights stop to change. Thus, it seems that the following conclusions can be drawn: the previous change of the weight can be used while determining the next change (and some algorithms really use it, for example in the form of momentum), the changes of particular weights can differ significantly, the weights that are no longer changing can be either frozen or pruned.

If we know the typical tendencies, then we can try to use some educated guesses of the weight values in the next epoch. Frequently the verification of the guess is quicker than calculating the value from scratch. Moreover, there are strong differences between the changes of weights in particular layers, however, the differences depend on the training algorithm. Detailed discussion of the weight changes for backpropagation, Levenberg-Marquardt algorithm, numerical gradient and VSS algorithm can be found in chapters 2.3 and 2.4.

1.5. Neural Activity and Data Spaces

While calculating the network error values the input signals are given to the first layer of the network and then the signals propagate through the network layer by layer. The number of signals *i(n)* propagating simultaneously in parallel through the *n-th* layer equals to the number of this layer outputs, what in turn equals to the dimensionality of this layer "signal space" or "data space" or "data representation" or "hypercube". The first space is the input data space (feature space), then there are as many hidden spaces as the number of hidden layers (in practice 0, 1 or 2) and finally there is the output (class) space.

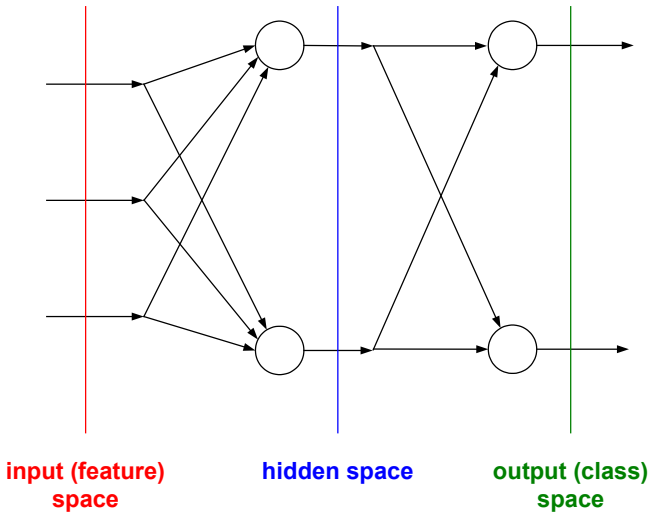

Fig. 1.42. Data spaces in a three-layer network.

The aim of this chapter is to analyze how particular vectors of the training or test set are placed in all the data spaces, how their positions change during the training and to draw some practical conclusions from this analysis. A single layer of a network can correctly divide only data, which can be separated by a single hyperplane (linearly separable data). If the network has more layers than each next layer separates with hyperplanes the data space of the previous layer.

In classification problems, the training data is divided into labeled subsets corresponding to classes. Neural networks try to map each of the training subsets into one of the vertices of the hypercube created in the output space. The task of the hidden layers is to map the vectors from the feature space in such a way that they could be separated according to their classes with the hyperplanes determined by the output layer. The higher layer frequently simplifies the internal representation of the lower layer by reducing the dimensionality of the data space and by reordering the training vectors (Fig. 1.43). The input data can be visualized in the input hypercube, the representation of hidden layers in their hypercubes and finally the network output in the output hypercube. If the dimensionality of a given hypercube is higher than three, then it is more practical to use parallel coordinates, though also other projection methods can be used [Duch 2004a].

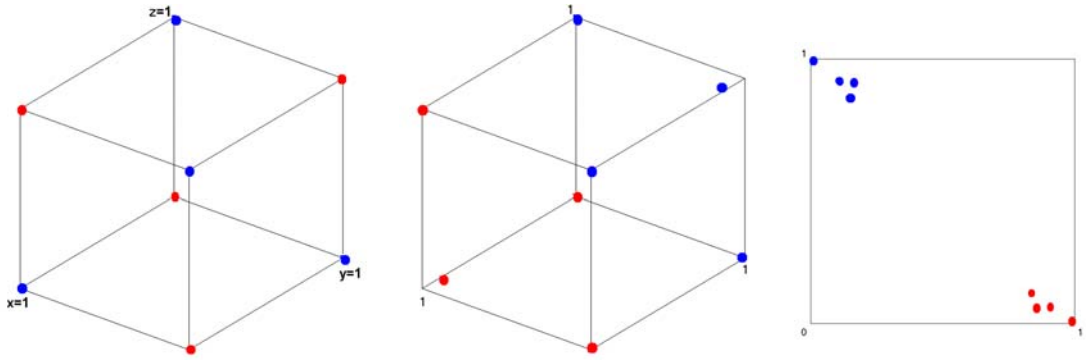

Fig. 1.43. Vectors of 3-bit parity (3-3-2) in the hypercubes of feature, hidden and class spaces.

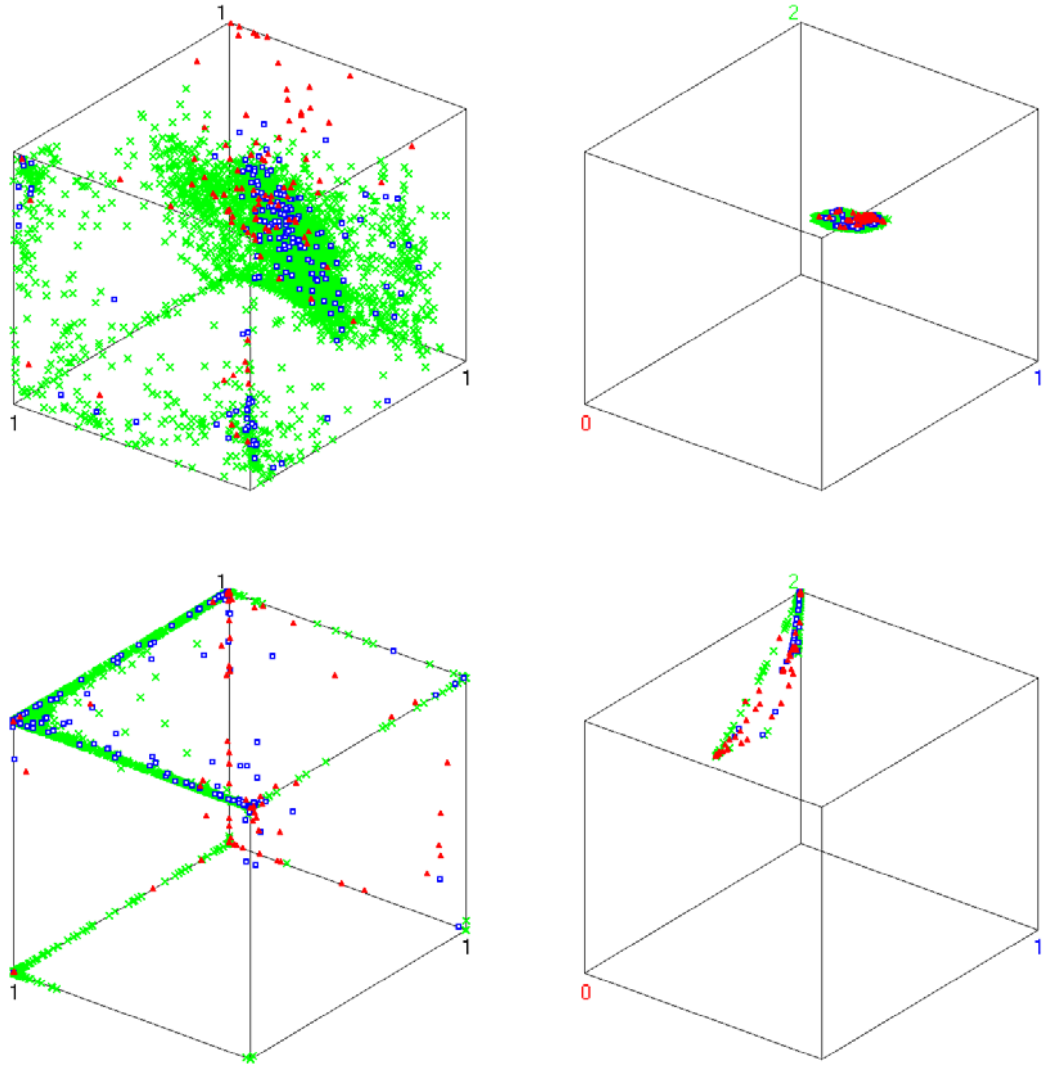

Fig. 1.44. Hidden (left) and output (right) neuron signals for Thyroid (21-3-3). Top: before the training starts (random initial weights). Bottom: after the 1st training cycle of VSS.

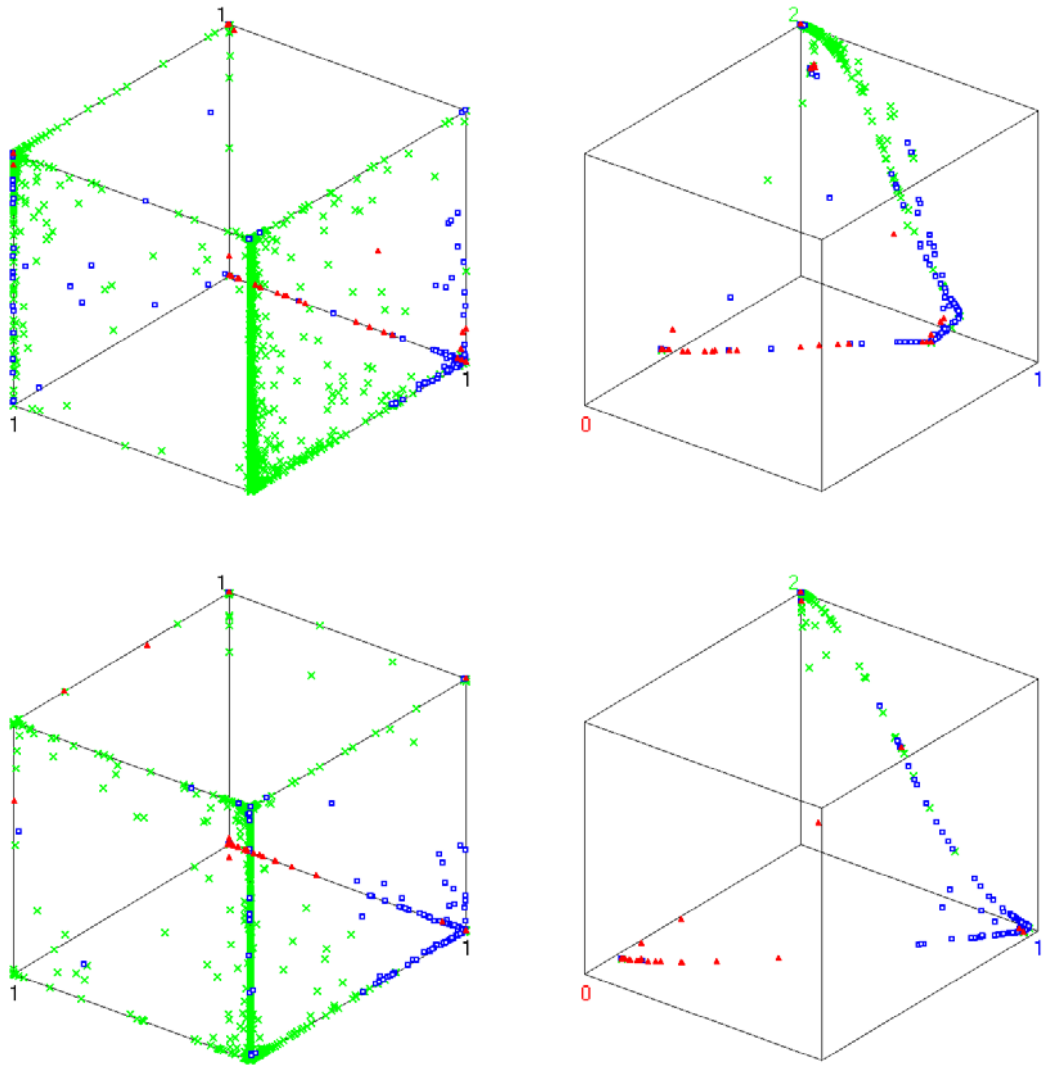

Fig. 1.45. Hidden (left) and output (rights) neuron signals for Thyroid (21-3-3). Top: after the 4^{th} training cycle of VSS. Bottom: after the 20^{th} training cycle of VSS.

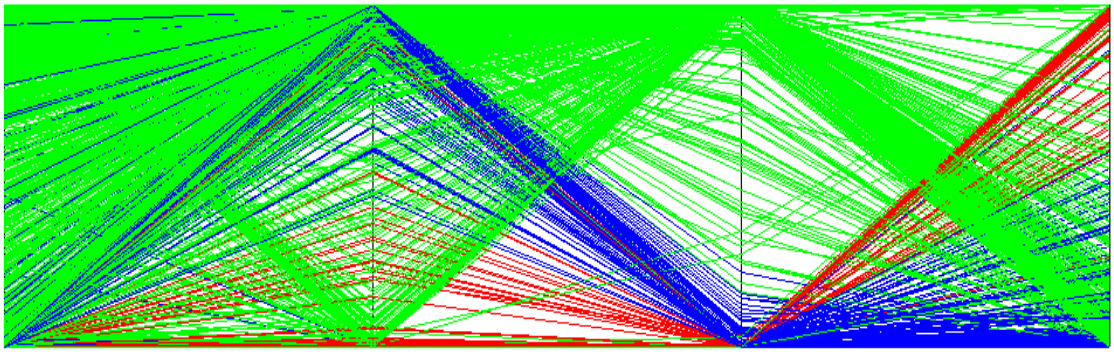

Fig. 1.46. Hidden neuron signals for Thyroid (21-4-3) after the 10^{th} training cycle of VSS shown in parallel coordinates.

Neural networks can achieve the same results using various weights. With the same distribution of vectors in the feature and class spaces, there exist many vector distributions in hidden spaces. The various possible distributions are visible as symmetries in the error surface plots.

Some vectors are far from the decision borders (close to the proper vertices in the hypercube). These vectors do not provide useful information in the training process, since they give almost zero error. An easy and effective method to accelerate the network training is to gradually eliminate or group together some of the vectors [Duch 2004b] (chapter 2.5.1).

1.6. Standard and Balanced Classification Accuracy

The Thyroid dataset (chapter 3.2.13.5) is an example of a dataset with an asymmetric class distribution. The training set has 3772 vectors, 93 of class 1, 191 of class 2 and 3488 of class 3. The test set has 3428 vectors, 73, 177 and 3178 of class 1,2 and 3 respectively. Thus the percentage of vectors in particular classes is 2.47%, 5.06% and 92.47% for the training set and 2.13%, 5.16% and 92.71% for the test set. In such a situation it is likely that the cost of misclassifying a vector of class 1 as a vector of class 3 will be higher than vice versa (chapter 3.2.12.1).

Frequently the network training on datasets with unbalanced classes is more difficult because big flat ES areas (situated in the front part of Fig. 1.49-left) corresponding to the majority classification accuracy are difficult to leave (see chapter 1.2.5.2).

The standard accuracy is given by

$$A_{std} = \frac{correct}{total} \tag{1.24}$$

where *correct* is the number of correctly classified vectors and *total* is the total number of vectors in the dataset.

The balanced (weighted) classification accuracy is defined here by

$$A_{bal} = \frac{1}{nc}\sum_{c=1}^{nc}\frac{correct(c)}{total(c)} \tag{1.25}$$

where *nc* is the number of classes, *correct(c)* is the numbers of correctly classified vectors of class *c* and *total(c)* is the total number of vectors in class *c*.

A network training can be optimized for standard or for balanced accuracy by adjusting the error function. With a square error function, the standard error is given by

$$E_{std} = \sum_{c}\sum_{v}(d_{v,c} - s_{v,c})^2 \tag{1.26}$$

and the balanced error is given by

$$E_{bal} = \frac{total}{nc} \sum_{c} \left[\frac{1}{total(c)} \sum_{v} (d_{v,c} - s_{v,c})^2 \right] \tag{1.27}$$

where *d* is the desired signal, *s* is the observed signal of an output layer neuron *c* in response to vector *v* and *total(c)* is the total number of vectors in class *c*.

A network trained with the error function (1.26) achieves higher standard accuracy and with (1.27) - higher balanced accuracy. Also the error surface of both networks looks differently, since among other factors, the error surface depends on the error function. In the first case the PCA-based ES projection shows asymmetries, which are caused by the unequal class distribution. In the second case the error surface projection becomes symmetric because the error function (1.27) has an equivalent influence on the error surface as balancing the number of instances in each class of the training set.

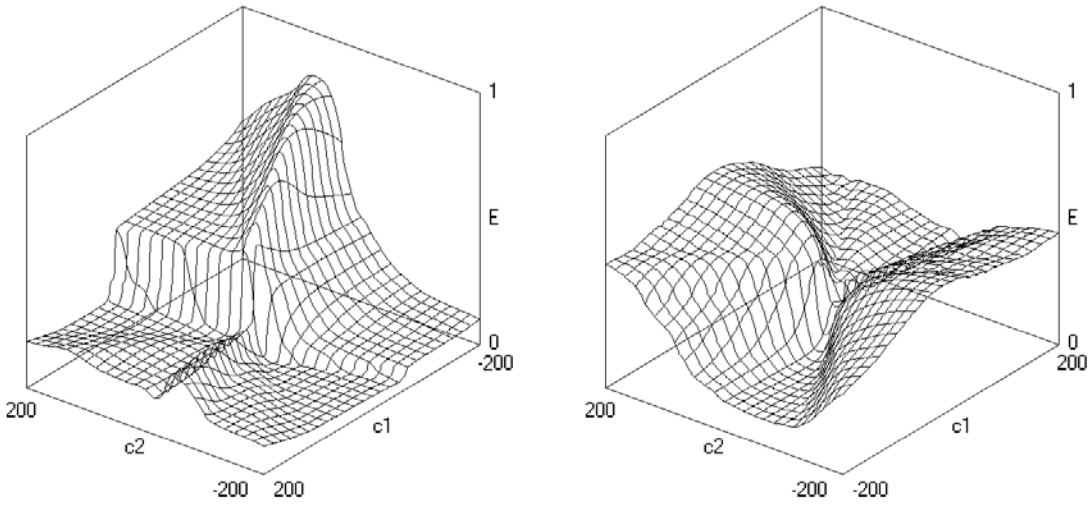

Fig. 1.47. Projection of Thyroid (21-4-3) error surface in the first and second PCA direction obtained with: left - standard error function (1.26), right - balanced error function (1.27).

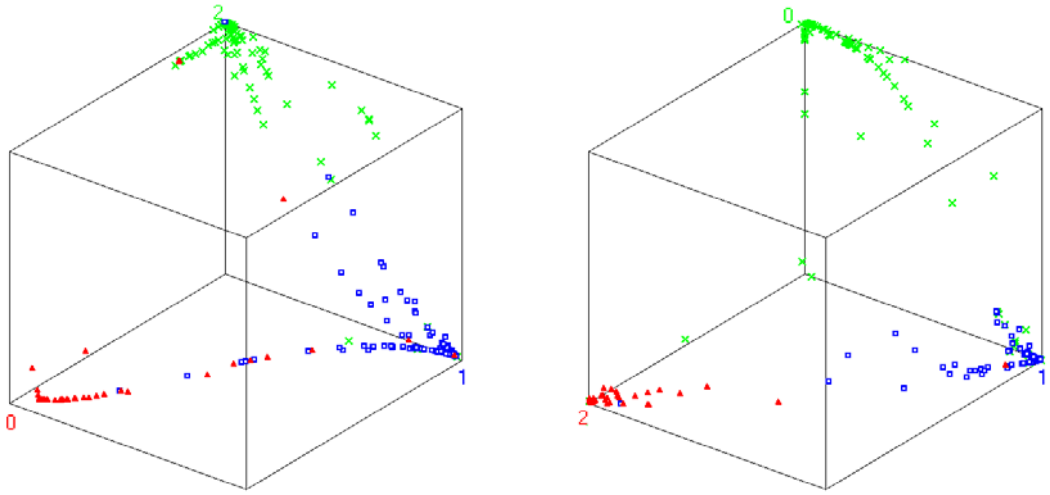

Fig. 1.48. Output neuron signals for Thyroid (21-4-3) after the 10^{th} training cycle. Left: with standard error function, right: with balanced error function (visible better separation of classes with fewer vectors).

Table. 1.3. Classification accuracies for thyroid dataset with standard and balanced error functions achieved after 10 cycles of VSS training. (Longer training allows for much higher accuracies - see chapter 3.2.12.5).

error function	E_{std}	E_{bal}
A_{std}(%training)	99.39	99.34
A_{bal}(%training)	95.30	99.60
A_{std}(%test)	98.13	98.02
A_{bal}(%test)	85.17	92.98

1.7. Decision Borders

MLP decision borders are hypersurfaces in the feature space that divide the space into subspaces assigned to particular classes. After the network training is finished, the vectors in the class space should be situated close to these hypercube vertices, which correspond to their classes. Vectors situated on the decision borders in the feature space will be placed on the equidistance hypersurfaces (shown in gray in Fig.1.49-right) in the class space.

In the example shown in Fig. 1.49. only two features (petal-width and petal-length) of the Iris dataset were used for network training (the network structure was 2-2-3). The test set consisted of 961 vectors (31 rows and 31 columns), which evenly covered all the space in Fig.1.49-left. That allowed for determining the decision borders, which are shown in Fig.1.49-left. The representation of the test vectors in the class space is shown in Fig. 1.49-right.

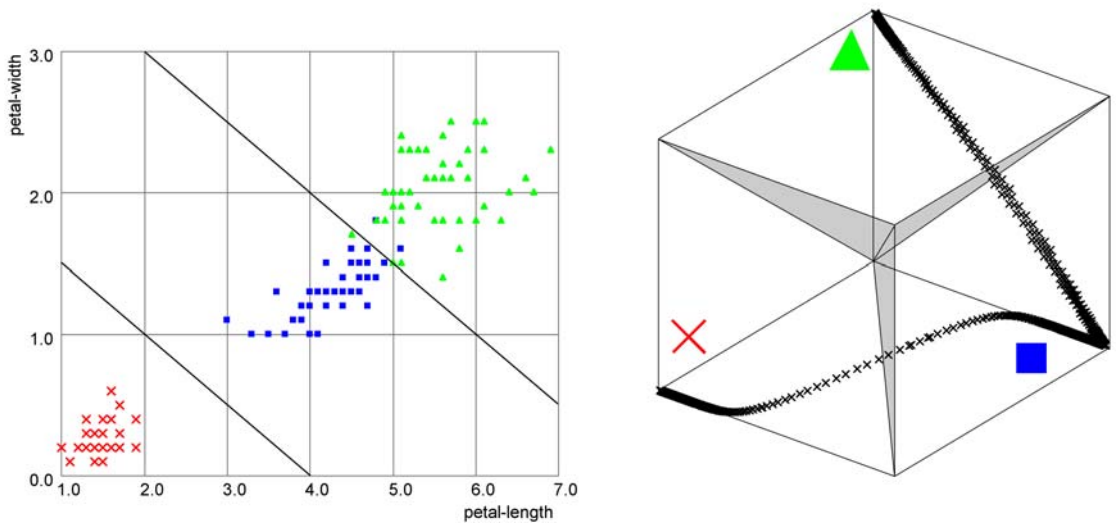

Fig. 1.49. Left: training vectors and decision borders of Iris (2-2-3) using only two most significant features. Right: class subspaces and representation of test vectors in the output space.

It is an interesting observation that although the test vectors cover evenly the entire input space, they do not cover the entire output space, but most of them is attracted to the proper vertices and almost all others are located close to the lines connecting the vertices of two neighbor classes. The same was observed for the Thyroid dataset (Fig. 1.45, 1.48). The classes represented by the red cross and blue square are separated quite clearly. On the other hand the border between the vectors shown in green and the vectors shown in blue is not so sharp, with higher density of the test vectors in the intermediate region.

Frequently a great disadvantage of neural networks is that as a result of a little perturbation of the input values the vector is classified to a different class [Duch 2005]. Most of the vectors in the output space are concentrated close to the hypercube vertices and do not reflect their probability of particular class membership. Thus, it may be desired to provide more smooth transitions between the vertices of the output hypercube. One possible way to do it is to use weight regularization or early stopping of the network training (chapter 2.6.2). Although such a network may have lower training classification accuracy, it can provide more information, its decisions will be more stable and it may generalize better avoiding overfitting of the data. Thus, the classification accuracy, frequently used as the only measurement of the classifier quality, is not the only value that should be taken into consideration (chapter 3.2.12.1). Decision borders will be further considered in chapter 2.6.2.

In the same way, as training the neural network is equivalent to searching for a minimum on the error surface, extracting logical rules from data is equivalent to providing the description of decision borders (chapter 3.2).

Part 2

Search-based Algorithms for MLP Training

2.1. Review of MLP Training Algorithms

2.1.1. Analytical Gradient-based Algorithms

2.1.1.1. Backpropagation

Backpropagation was the first successful training algorithm for multilayer perceptron [Werbos 1974][Rumelhart 1986]. Other analytical gradient-based algorithms use the same error backpropagation mechanism, but different weight update methods.

The sum-squared error function, which is minimized by backpropagation algorithm, can be written in the following way:

$$E(w_{ij}^{(n)}) = \frac{1}{2}\sum_{v}\sum_{j}(desired_j - out_j^{(n)})^2 \tag{2.1}$$

where $desired_j$ is the desired signal and out_j is the actual signal of the *j-th* output neuron. The error is summed over all *j* output neurons and all *v* vectors. The network weights are adjusted by a series of gradient descent updates. For sigmoid transfer function after some calculations that can be found literature, the equations that constitute the basic BP algorithm are obtained in the form presented below. We define

$$delta_k^{(n)} = (desired_k - out_k^{(n)}) \cdot out_k^{(n)} \cdot (1 - out_k^{(n)}) \tag{2.2}$$

as the delta for the output layer, where *n* is the index of the layer. Then we back-propagate the deltas to earlier layers using

$$delta_k^{(n)} = \left(\sum_k delta_k^{(n+1)} \cdot w_{lk}^{(n+1)} \right) \cdot out_k^{(n)} \cdot (1 - out_k^{(n)}) \tag{2.3}$$

where w_{kl} is the weight connecting the *k-th* neuron in the *n-th* layer with the *l-th* neuron in the $n+1$ layer. Then each weight update equation can be written as

$$\Delta w_{kl}^{(n)} = \eta \sum_v delta_k^{(n)} \cdot out_j^{(n-1)} \tag{2.4}$$

To enhance the BP algorithm, variable learning rate and momentum can be used. Since similar enhancements can be used with numerical gradient (NG), they will be described in detail in the chapters about NG.

2.1.1.2. RPROP (Resilient Backpropagation)

RPROP is a modification of standard backpropagation, which considers only the sign of the derivative, but not its value [Riedmiller 1992]

$$\Delta w_{ij}(k) = -\eta_{ij} \operatorname{sgn}\left(\frac{\partial E(k)}{\partial w_{ij}(k)} \right) \tag{2.5}$$

The learning rate η is adjusted individually for each weight in each training cycle; if the direction of error derivative with respect to w_{ij} is the same in the actual and in the previous epoch then

$$\eta_{ij}(k) = \min(a\,\eta_{ij}(k-1), \eta_{\max}) \tag{2.6}$$

where a and b are constants, a=1.2, b=0.5.
If the directions of error derivatives with respect to w_{ij} in the present and in the previous epochs are opposite then

$$\eta_{ij}(k) = \max(b\,\eta_{ij}(k-1), \eta_{\min}) \tag{2.7}$$

If the direction of error derivative with respect to w_{ij} was zero either in the actual or in the previous epoch then

$$\eta_{ij}(k) = \eta_{ij}(k-1) \tag{2.8}$$

Thus, the learning rate increases if the derivative sign in two successive epochs is the same and decreases otherwise. RPROP due to omitting the information about gradient value makes the learning process much faster in the areas of low error surface steepness.

2.1.1.3. Quickprop

The idea of the quickprop algorithm [fahlman88][Osowski 1996][Duch 2000] is to approximate the minima on the error surface with a parabola. Using the values of weights and gradients in two points (β in equation 2.11) a parabola is determined and a step is made to its minimum.

Quickprop algorithm uses the following rule of weight changes:

$$\Delta w_{ij}(k) = -\eta \left(\frac{\partial E(k)}{\partial w_{ij}(k)} + \gamma w_{ij}(k) \right) + \alpha_{ij}(k) \Delta w_{ij}(k-1) \tag{2.9}$$

The change of the weight depends here on three factors: the error function derivative with respect to this weight, the actual value of the weight and the previous change of the weight (momentum). The coefficient γ (typically $\gamma \approx 10^{-4}$) is responsible for weight reduction and prevents the weights from excessive growth. The learning rate η takes one of the two values:

$$\eta = \eta_0 \quad \text{if} \quad \left(\frac{\partial E(k)}{\partial w_{ij}(k)} + \gamma w_{ij}(k) \right) > 0 \quad \text{or} \quad \Delta w_{ij}(k-1) = 0 \tag{2.10}$$

otherwise $\eta = 0$

The momentum term α is adjusted individually for each weight in each epoch:

$$\alpha_{ij}(k) = \alpha_{\max} \quad \text{if} \quad (\beta_{ij}(k) > \alpha_{\max} \quad \text{or} \quad S_{ij}(k) \Delta w_{ij}(k-1) \beta_{ij}(k) < 0) \tag{2.11}$$

otherwise $\alpha_{ij}(k) = \beta_{ij}(k)$

where $S_{ij}(k) = \frac{\partial E(k)}{\partial w_{ij}(k)} + \gamma w_{ij}(k)$ and $\beta_{ij}(k) = \frac{S_{ij}(k)}{S_{ij}(k-1) - S_{ij}(k)}$

Quickprop is much quicker than standard backpropagation and less prone to spurious local minima. Also a simpler version of the algorithm exists, which uses only the parabolic approximation to find the error function minimum.

2.1.1.4. Scaled Conjugate Gradient

With initial gradient g_0 and initial vector $p_0 = -g_0$ the conjugate gradient method recursively constructs two vector sequences:

$$g_{i+1} = \nabla E(w_{i+1}) \quad \text{and} \quad p_k = -g_k + \beta_k p_{k-1} \tag{2.12a}$$

where $\beta_k = \frac{|g_k|^2 - g_k^T g_{k-1}}{|g_{k-1}|^2}$ (also other definisions of β_k a can be used)

where g is gradient direction and p is called conjugate direction. We proceed from w_i along the direction p_i to the minimum of E at w_{i+1} through line minimization and then set g_{i+1} at the minimum.

The idea of the conjugate gradient is to spoil the results of the previous step in the current step as little as possible by making the current step in the direction orthogonal to the previous step. The conjugate direction p minimizes trajectory oscillations and allows longer steps, which leads to a faster convergence than steepest descent directions, although the error function decreases most rapidly in the steepest descent directions.

Scaled conjugate gradient algorithm is a version of conjugate gradient that avoids the time-consuming line search along conjugate directions. SCG algorithm [Möller 1993][Haykin 1994] is considered to be the quickest one among the well-known algorithms for larger networks. As a Levenberg-Marquardt algorithm, it introduces a scalar λ to regulate the Hessian $H = \nabla^2 E$.

$$\Delta w_k = \varepsilon_k p_k \tag{2.12b}$$

$$\text{step size:} \quad \varepsilon_k = \frac{p_k^T g_k}{p_k^T s_k + \lambda_k \mid p_k \mid^2} \tag{2.12c}$$

$$s_k = \frac{g(w_k + \sigma_k p_k) - g_k}{\sigma_k} \tag{2.12d}$$

$$\Delta_k = \frac{E(w_k) - E(w_k + \varepsilon_k p_k)}{E(w_k) - E_q(w_k + \varepsilon_k p_k)} \tag{2.12e}$$

where E is the real error, E_q is the quadratic approximation of the error, λ_k is a scaling factor, which in each iteration is raised or lowered according to how close E/E_q is to one, i.e. how close the error approximation is to the real error.

Since conjugate gradient methods do not compute any matrices, they scale well with the network size (chapter 2.4.5).

2.1.1.5. Quasi-Newton

Newton's method is an alternative to the conjugate gradient methods for fast optimization. [Dennis 1983] [NN Tolbox 2004]. The gradient descent algorithm uses the following update rule:

$$w_{i+1} = w_i - \lambda_0 \nabla E(w_i) \tag{2.13}$$

Expanding the gradient of $E(w)$ using a Taylor series around a point w_i,

$$\nabla E(w_{i+1}) = \nabla E(w_i) + (w_{i+1} - w_i)^T \nabla^2 E(w_i) + rest\,, \tag{2.14}$$

Solving the equation $\nabla E(w_i) = 0$ and neglecting the higher order *rest*, we get Newton's update rule:

$$w_{i+1} = w_i - \frac{\nabla E(w_i)}{H(w_i)} \tag{2.15a}$$

where $H(w_i) = \nabla^2 E(w_i)$ is the Hessian matrix of the performance index at the current values of the weights and biases.

$$\nabla E = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_n} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial e_{1P}}{\partial w_1} & \frac{\partial e_{1P}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{MP}}{\partial w_1} & \frac{\partial e_{MP}}{\partial w_2} & \cdots & \frac{\partial e_{MP}}{\partial w_n} \end{bmatrix} \qquad H = \begin{bmatrix} \frac{\partial E}{\partial w_1^2} & \frac{\partial E}{\partial w_2 \partial w_1} & \cdots & \frac{\partial E}{\partial w_n \partial w_1} \\ \frac{\partial E}{\partial w_1 \partial w_2} & \frac{\partial E}{\partial w_2^2} & \cdots & \frac{\partial E}{\partial w_n \partial w_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial w_1 \partial w_n} & \frac{\partial E}{\partial w_2 \partial w_n} & \cdots & \frac{\partial E}{\partial w_n^2} \end{bmatrix} \tag{2.15b}$$

Because it is complex and expensive to compute the Hessian matrix ***H*** for feedforward neural networks, a version of the algorithm called quasi-Newton (or secant) was developed. It updates only the approximate Hessian matrix ***G*** entries at each iteration *k* of the algorithm. The most pupular method of calculating the approximate Hessian G is the BFGS (Broyden-Goldfarb-Fletcher-Shanno) method, which calculates the inverse of the approximate Hessian:

$$V_k = V_{k-1} + \left[1 + \frac{r_k^T V_{k-1} r_k}{s_k^T r_k} \right] \frac{s_k s_k^T}{s_k^T r_k} - \frac{s_k r_k^T V_{k-1} + V_{k-1} r_k s_k^T}{s_k^T r_k} \tag{2.15c}$$

where $V_k = G_k^{-1} \qquad V_0 = 1 \qquad s_k = W_k - W_{k-1} \qquad r_k = \nabla E_k - \nabla E_{k-1}$

Quasi-Newton algorithm requires more computation in each iteration and more storage than the conjugate gradient methods, although it generally converges in fewer iterations. The approximate Hessian must be stored, and its dimension is N_w x N_w, where N_w is equal to the number of weights and biases in the network. For very large networks it may be better to use Rprop or the conjugate gradient algorithm. For smaller networks, however it can be an efficient training method.

2.1.1.6. Levenberg-Marquardt Algorithm

LM algorithm is considered to be the quickest one from the well-known algorithms for smaller networks [Ranganathan 2004][Marquardt 1963][Ranga 2004][Fang 1999]. LM algorithm uses both gradient descent and curvature information (Newton's method). Combining these two algorithms, the following update rule can be written:

$$w_{i+1} = w_i - (H + \lambda I)^{-1} \nabla E(w_i) \tag{2.16}$$

where $H = \nabla^2 E(w_i)$ is the Hessian matrix and $\nabla E(w_i)$ is the Jacobian matrix. Replacing the identity matrix with the Hessian diagonal increases the step in the direction of small gradient minimizing the trajectory oscillations. Thus, we get the final Levenberg-Marquardt update rule:

$$w_{i+1} = w_i - (H + \lambda diag[H])^{-1} \nabla E(w_i) \tag{2.17}$$

λ is dynamically decreased about a rank of order if the error decreases. If the error increases λ is increased about a rank of order, learning trajectory returns to the previous point and the step is repeated.

The main disadvantage of LM algorithm is a high memory requirement. The size of Jacobian is $N_v N_o N_w$ and that of Hessian is N_w^2. That in practice terms means that for satellite image database with 27 hidden neurons as discussed in [Ranga 2004] the Jacobian alone requires 248 MB memory using double (8 Byte) type. To reduce the memory requirements the Jacobian may be divided into several parts and Hessian calculated by summing partial results but this adds a significant computational overhead. For comparison the VSS algorithm (chapter 2.4) requires $N_v(N_o + N_h) + 2N_w$ memory for network parameters, what gives only 1.26 MB memory with double (8 Byte) type. Storing the training set in memory requires 1.30 MB with double type. (N_w is the number of weights, N_v – number of training vectors, N_h – number of hidden neurons, N_o – number of output neurons.)

2.1.1.7. RLS

The RLS (Recursive Least Square) algorithm relies on the analogy between adaptive filters and neural networks [Azimi 1992][Bilski 2002, 2004]. It is well known that in adaptive filtering the RLS algorithm is typically an order of magnitude faster than LMS algorithm (on which backpropagation is based).

The algorithm minimizes the following performance measure:

$$J(n) = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} e_j^{L^2}(t) \tag{2.18}$$

where λ, called the forgetting factor, is a positive constant less than one, *e* is the error in the linear part of the *j*-th neuron in the highest (*L*-th) layer, *n* is the number of the iteration and *NL* is the number of neurons in the highest layer. The detailed equations describing RLS algorithm are rather complex and therefore even though RLS requires fewer training cycles than BP the total computational cost of RSL is comparable or only slightly lower than that of BP.

2.1.2. Global Optimization Algorithms

Global minimization methods applied to neural networks are computationally much more costly than gradient-based methods. Nevertheless, they are used because of their ability to find frequently much better solutions then analytical gradient-based methods can find [Duch 1999a].

2.1.2.1. Simulated Annealing

Simulated Annealing [Kirkpatrick 1983] is inspired by the annealing (cooling) process of crystals that reach the lowest energy corresponding to perfect crystal structure, if cooled significantly slowly.

An annealing methodology requires three functions [Harold 1997]: the probability distribution of parameters

$$G_T(x+X) = (2\pi T)^{-0.5} \exp(-X^2/T) \tag{2.19}$$

the probability of accepting the new set of parameters, based on the energy landscape property at the new and at the old states

$$P_T = 1/[1+\exp(-\Delta E/T)] \tag{2.20}$$

and the cooling schedule for changing the temperature T to generate a new state

$$T = T_0/(1+T) \tag{2.21}$$

In an interesting paper by Engel [Engel 1998], simulated annealing was applied to a network in which the adaptive parameters were discretized.

2.1.2.2. Alopex

Alopex uses local correlations between changes in individual weights and changes in the global error measure [Unnikrishnan 1994]. The algorithm is stochastic and uses the temperature parameter in a manner similar to that in simulated annealing.

The algorithms can be described as follows: consider a neuron i connected to neuron j with a weight w_{ij}. During the n^{th} iteration, the weight w_{ij} is updated according to the rule:

$$w_{ij}(n) = w_{ij}(n-1) + \delta_{ij}(n) \tag{2.22}$$

where $\delta_{ij}(n) = -\delta$ with the probability $p_{ij}(n)$ and $\delta_{ij}(n) = \delta$ with the probability $1-p_{ij}(n)$. The probability for the negative step $p_{ij}(n)$ is given by the Boltzman distribution:

$$p_{ij}(n) = 1/[1 + \exp(C_{ij}(n)/T(n))] \tag{2.23}$$

where $C_{ij}(n)$ is given by the correlation

$$C_{ij}(n) = \Delta w_{ij}(n) \cdot \Delta E(n) \tag{2.24}$$

and $T(n)$ is positive temperature. $\Delta w_{ij}(n)$ and $\Delta E(n)$ are the changes of weight w_{ij} and the error measure E over the previous two iterations.

$$\Delta w_{ij}(n) = w_{ij}(n-1) - w_{ij}(n-2) \tag{2.25}$$

$$\Delta E(n) = E(n-1) - E(n-2)$$

If $\Delta E(n)$ is negative, the probability of moving each weight in the same direction is greater than 0.5. The temperature T is updated every N iterations using the following annealing schedule:

$$T(n) = \frac{\delta}{N} \sum_{n'=n-N}^{n-1} |\Delta E(n')| \text{ if } n \text{ is a multiple of } N \text{ and } T(n)=T(n-1) \text{ otherwise} \tag{2.26}$$

In Alopex, the magnitude of Δw is the same for all weights and that point does not seem to be a very good solution since it does not take an advantage of the natural ill-conditioning of MLP error surfaces. So in empirical tests [Unnikrishnan 1994] Alopex required 487 training cycles to solve the Xor problem, while algorithms such as Levenberg-Marquardt or proposed further in this thesis Variable Step Search Algorithm require less than 10 training cycles for the Xor problem. On the other hand, the aim of global optimization algorithms is not to compete with local algorithms for the speed but for the quality of solution for difficult problems. Thus, it seems worthwhile to modify Alopex so that it could use different Δw for different weights.

2.1.2.3. NOVEL Algorithm

Novel is a hybrid, global-local trajectory-based method, exploring the solution space, locating promising regions and using local search to locate promising minima [Shang 1996]. Trajectory in the global search stage is defined by a differential equation

$$\dot{P}(t) = -\mu_g \nabla M(P(t)) + \mu_l (T(t) - P(t)) \tag{2.27}$$

The first component allows the trajectories to be attracted by local minima, and the second one allows them to walk out of the minima. The trace function T should assure that all space is finally traversed. It may either partition the space into regions or make first coarse and then fine search.

2.1.2.4. Genetic Algorithms

Genetic Algorithms (GA) were proposed by Holland [Holland 1992] in the 1970s as an algorithmic concept based on a Darwinian-type survival-of-the-fittest strategy with sexual reproduction, where stronger individuals in the population have a higher chance of creating an offspring [Rutkowska 1997][Jain 1998][Michalewicz 2003].

Every member of a population has a certain fitness value associated with it, which represents the degree of correctness of that particular solution or the quality of solution it represents. The basic approach is to model the possible solutions to the search problem as strings of ones and zeros. The strings are manipulated by the GA using genetic operators, to finally arrive at a quality solution to the given problem. Although GA do not guarantee convergence to the single best solution to the problem, they are frequently efficient search techniques. The main advantage of GA is that they are able to manipulate numerous strings simultaneously, where each string represents a different solution to a given problem. Thus, the possibility of the GA getting stuck in local minima is greatly reduced because the whole space of possible solutions can be simultaneously searched. A basic genetic algorithm comprises three genetic operators:

- selection
- crossover
- mutation

Starting from an initial random population of strings (representing possible solutions), the GA use these operators to calculate successive generations. First, pairs of individuals of the current population are selected to mate with each other to form the offspring, which then forms the next generation. Selection is based on the survival-of-the-fittest strategy with the key idea to select the better individuals of the population. The most commonly used strategy to select pairs of individuals is the method of roulette-wheel selection, in which every string is assigned a slot in a simulated wheel sized in proportion to the string's relative fitness. This ensures that highly fitted strings have a greater probability to be selected to form the next generation through crossover. The mutation operator, which with low probability randomly changes single bits in the individuals, is introduced to prevent premature convergence into a suboptimal solution. After selection of the pairs of parent strings, the crossover operator is applied to each of these pairs.

The crossover operator involves the swapping of genetic material (bit-values) between the two parent strings. In a single point crossover, a bit position along the two strings is selected at random and the two parent strings exchange their genetic material as illustrated below.

Parent A = a1 a2 a3 a4 | a5 a6 Parent B = b1 b2 b3 b4 | b5 b6 (2.28)

The swapping of genetic material between the two parents on either side of the selected crossover point, represented by "|", produces the following offspring:

Offspring A= a1 a2 a3 a4 | b5 b6 Offspring B= b1 b2 b3 b4 | a5 a6 (2.29)

Genetic algorithms are very popular as a training method for neural networks [Matthews 2000][Seiffert 2001], although not many commercial programs use them and it is difficult to find results that show their clear advantage in this type of applications.

In the case of neural network training with GA, the fitness function corresponds to the network error (e.g. MSE), and particular weight values are encoded in the genome. However, the fitness function can be also a weighted sum of network error and network complexity, both encoded in the genome [Kwaśnicka 2004]. The networks can be also trained with gradient-based methods and GA can be used only for optimization of network topology - then genome encodes only the network structure [Mandischer 1993]. Genetic algorithms can also be used as one possible method of SMLP network training (chapter 3.2.10).

2.2. Basis of Search Algorithms

Search is a systematic examination of states to find a path from the start state to the goal state. The output of a search algorithm is a solution to the problem. The basic search algorithms can be divided as follows:

Uninformed (blind) search methods:
- Depth-First
- Breadth-First

Informed (heuristics) search methods:
- Beam-Search
- Hill Climbing
- Best-First

The simplest search methods are uninformed. They have no information about the state space and perform blind systematic search.

If knowledge about the problem is available, we can attempt to guide the search to a more efficient conclusion. The knowledge we have about the solution cannot be explicit - this would mean we could solve the problem directly. Instead, we have rules of thumb – heuristics. They are not guaranteed to find a good solution, nor necessarily to find one at all, but they will usually help us find an adequate solution more swiftly.

2.2.1. Depth-First Search

The depth-first search algorithm searches through the tree systematically, exploring each branch until it finds a goal node. One alternative is selected and pursued at each node until the goal is reached or a node is reached where further downward motion is impossible. When further downward motion is impossible, the search is restarted at the nearest ancestor node with unexplored children. This search is complete and non-optimal (the algorithm will

not necessarily find the most efficient route through the state space). For a tree with branching factor *b* (average number of children of each node) at depth *d*, the time efficiency is O(*bd*) and the space efficiency is O(*bd*).

The Depth-First Algorithm:

1. Form a one element queue *Q* consisting of the root node
2. Until *Q* is empty or the goal has been reached, determine if the first element in *Q* is the goal
    a. If it is, do nothing
    b. If it is not, remove the first element from *Q* and add the first element's children, if any, to the front of *Q*
3. If the goal is reached then success else failure

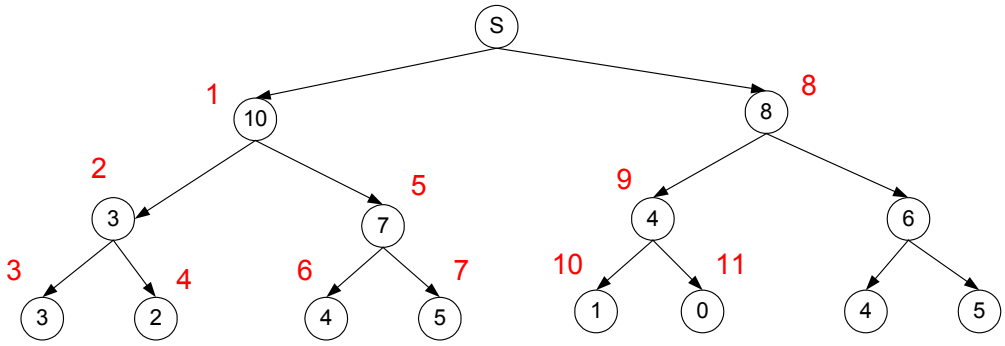

Fig. 2.1. The Depth-First Search. The numbers inside the nodes correspond to the error value. The red numbers outside the nodes show the order in which the nodes were examined.

2.2.2. Breadth-First Search

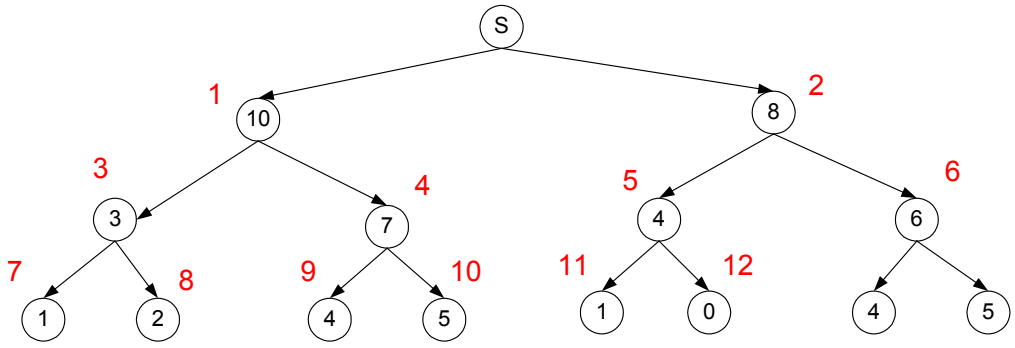

Fig. 2.2. The Breadth-First Search. The numbers inside the nodes correspond to the error value. The red numbers outside the nodes show the order in which the nodes were examined.

The breadth-first search algorithm searches for the goal node among all the nodes of a particular generation (level) before expanding further. If there is more than one goal node, always the nearest one in a given generation is found. This search is complete and non-optimal. Time efficiency O(*bd*). Space Efficiency: O(*bd*).

The Breadth-First Algorithm:

1. Form a one element queue *Q* consisting of the root node
2. Until *Q* is empty or the goal has been reached, determine if the first element in *Q* is the goal. If it is, do nothing b. If it is not, remove the first element from *Q* and add the first element's children, if any, to the back of *Q*
3. If the goal is reached then success else failure

2.2.3. Hill Climbing Search

Hill climbing search is based on depth-first search. A heuristic is used to improve the search efficiency. At each step, it is estimated if one choice is likely to be better than another and the choices are ordered accordingly. This search is complete and non-optimal.

The Hill Climbing Algorithm:

1. Form a one element queue *Q* consisting of the root node
2. Until *Q* is empty or the goal has been reached, determine if the first element in *Q* is the goal.
    a. If it is, do nothing
    b. If is not, remove the element from *Q*, sort the first element's children, if any, by estimating remaining distance, and add this sorted list to the front of *Q*
3. If the goal is reached then success else failure

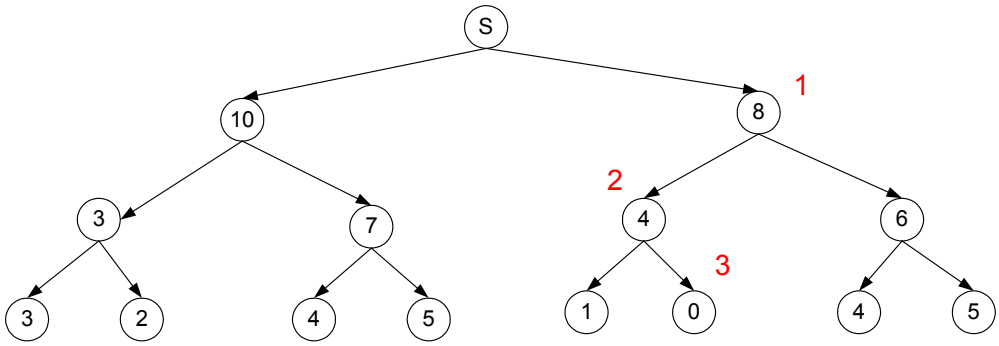

Fig. 2.3. The Hill Climbing Search. The numbers inside the nodes correspond to the error value. The red numbers outside the nodes show the order in which the nodes were examined.

2.2.4. Beam Search

Beam search avoids the combinatorial explosion problem of breadth first search by expanding only the *p* most promising nodes at each level. A heuristic is used to predict which nodes are likely to be closest to the goal. Beam search expands several partial paths and purge the rest. Beam search is like breadth-first search because it progresses level by level but it is also like depth-first search, because the beam search moves downward only through the

best p nodes at each level; the other nodes are ignored. This search is incomplete and non-optimal. There is a danger that a goal-finding route will be removed from Q before it can be explored. This may lead to not finding any goals at all. At each level there are only p nodes stored. This avoids the exponential explosion problem of breadth-first search.

The Beam Search Algorithm:

1. Form a one element queue Q consisting of the root node
2. Until Q is empty or the goal has been reached, determine if any of the elements in Q is the goal.
    a. If they are, do nothing
    b. If they are not, remove the elements from Q and add their children, if any, to the back of Q.
    c. Sort Q by heuristic.
    d. Remove all but the first p nodes from Q.
3. If the goal is reached then success else failure

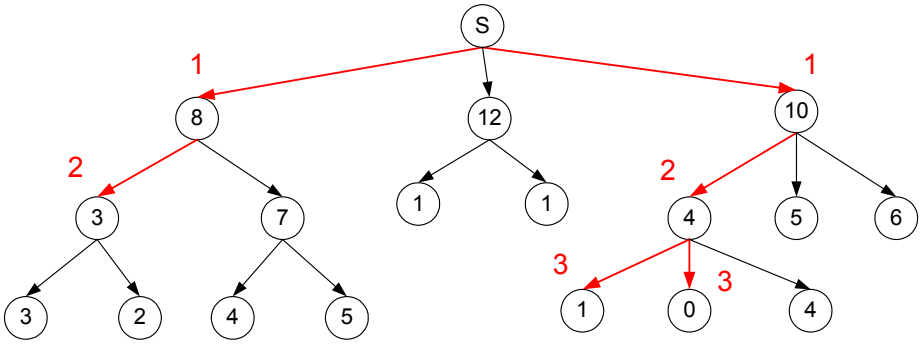

Fig. 2.4. The Beam Search. The numbers inside the nodes correspond to the error value. The red numbers outside the nodes show the order in which the nodes were examined and the red arrows the paths taken by the beam search.

2.2.5. Best-First Search

The Best-First Algorithm:

1. Form a one element queue Q consisting of the root node
2. Until Q is empty or the goal has been reached, determine if the first element in Q is the goal.
    a. If it is, do nothing
    b. If is not, remove the element from Q and add the first element's children to the Q.
    c. Sort Q by estimated remaining distance
3. If the goal is reached then success else failure

The Best-First Search is based on as well breadth- as on depth-first search. A heuristic is used to improve the search efficiency. At each step, the expansion of nodes is resumed from the most promising node opened so far, no matter where it is in the tree. This search is

complete and non-optimal. Since there are different ways to compute the most promising (best) node, there are some variants of the best-first search: uniform-cost search (estimated best is the least cost so far), greedy search (least estimated cost to goal), A* (cost so far plus estimated cost to goal), and many refinements of those.

2.2.6. Search Algorithms for MLP Training

There are two significant differences between the weight space in MLP networks and nodes of trees or graphs. First, network weights take continuous values. Second, except for very simple cases, it is impossible to determine the optimal value of a given weight in a single step and the process must be repeated iteratively always in interaction with other weights.

For that reason, the described above search algorithms are not very suitable for MLP networks and new search-based training algorithms, such as NG, VSS and SMLP training methods had to be developed.

If the weights of the network are thought of as tree nodes then the number of nodes is limited but each node can be assigned an infinite number of values. Also the points in the weight space can be thought of as tree nodes. Then there are an infinite number of nodes and each node can be assigned only two values (the learning trajectory passes or does not pass through this point).

The first approach, where a weight represents the node, is closer to SMLP networks (chapter 3.2), where the weights can take only three values (-1, 0, +1). SMLP training methods, which change one or two weights at a time, resemble the best-first search with many significant modifications. Also an SMLP training method based on the beam search is proposed.

The second approach, where a point in the weight space represents the node, is closer to the standard MLP networks. Numerical gradient (NG) and variable step search algorithm (VSS) use a strategy similar to hill climbing search.

However, it seems that there is no use to apply a modification of the beam search to standard MLP networks trained with NG and VSS. We can generate several sets of random weights (several starting points) but there are no forks in the road along the trajectory paths and all the beams converge to that one with the lowest error after the first training epoch.

Nevertheless, methods based on beam search can be used for MLP training, but not with such search methods, as NG or VSS. For example, an algorithm that makes a step in random directions instead of always downward can be implemented with beam search.

However, the node analogy does not seem to be the best choice for MLP networks and therefore it is no further used in the thesis. Instead, the weights and the points in the weight space are considered in MLP training algorithms.

2.3. Numerical Gradient

2.3.1. Overview of Numerical Gradient Algorithm

Numerical Gradient (NG) is a local gradient-based search algorithm. In contradiction to the training algorithms, which use analytical gradient, it does not require the knowledge of connection structure among neurons. Also the neural transfer functions do not have to be differentiable. Moreover, special tables that remember neuron signals can be used to reduce the computational cost up to several hundred times. In chapter 2.3 only the batch training is considered, the discussion of the semi-batch and on-line training can be found in chapter 2.5.2. The networks discussed here consist of usually three fully connected layers and the neurons use sigmoidal transfer functions. In the second part of the thesis it is assumed that the slope β of logistic sigmoids used as neural transfer functions equals one, thus the transfer function is given by the formula:

$$Y = \frac{1}{1 + e^{-u}} \tag{2.30}$$

where Y is the neuron output signal and u is given by (1.1). The networks considered here are used for data classification.

As all MLP learning algorithms, NG optimizes weights (including biases as w_0 weights) of output and all hidden layer neurons. Before the training starts, the weights are initialized with small random values. If the random initial weight range is too broad then it is a great chance that the starting point lies somewhere on the flat area of the error surface and as a result the network cannot be trained with any gradient-based or local search methods (chapter 1.2).

The initial values of all weights cannot be equal (e.g. all zero), because this would provide no difference between the signals of hidden neurons at the starting point. Although the gradient components are different from zero, they are the same for the corresponding weights of each hidden neuron, what makes the training impossible. This situation resembles the vertex of a cone, where the numerically calculated gradient components are different from zero, but they are the same in each direction and cancel each other, what finally gives zero gradient.

NG algorithm consists of two stages: finding the gradient direction and finding the minimal error along this direction. To find the gradient direction, a constant, small value dw is added to a single weight w and the error decrease $dE(w)$ is calculated as

$$dE(w) = E(w) - E(w + dw) \tag{2.31}$$

$E(w)$ takes the same value for all weights because the gradient component $dE(w)$ is calculated in the same point for each weight w.

Such a simple NG algorithm shows better convergence than standard backpropagation. NG without directional minimization converges better than BP without

directional minimization and NG with directional minimization better than BP with directional minimization. (In directional minimization the minimum along the gradient direction is searched for and than a step is made to that minimum.) NG in this version requires fewer training cycles than BP, but has higher computational effort per one training cycle. The total computational effort is comparable or frequently even higher than that of BP with optimal parameters. The modifications of NG, which reduce the computational cost and improve the algorithm convergence, will be successively introduced in the following chapters.

2.3.2. Signal Table

Since only one weight is changed at a time, the signals do not have to be propagated through the entire network to calculate the error, but only through the fragment of the network in which the signals are different before and after the change. The remaining signals incoming to all neurons of hidden and output layers are remembered for each training vector in an array called "signal table". With VSS the signals must be propagated through the entire network only once at the beginning of the training thus filling in the signal table and with NG once per each training cycle. The dimension of the signal table is N_V x (N_H+N_O) where N_V is the number of vectors in the training set and N_H and N_O the number of hidden and output neurons. After a single weight is changed, only the appropriate entries in signal table are updated. Also the error of each output neuron is remembered and does not have to be calculated again if a weight of another output neuron is changed. The signal table reduces three types of calculations: summing the signals incoming to the neuron, calculating the neural transfer function values and calculating the network error. It significantly shortens training times, especially for bigger networks. For a network structure 125-8-2 the training is accelerated about 35 times, for smaller networks less and for bigger networks more. The acceleration is stronger for VSS than for NG.

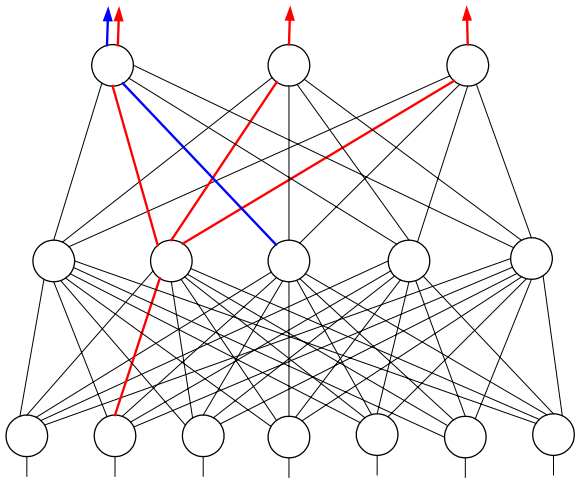

Fig. 2.5. Signals that change if an output neuron weight is changed are shown in red. Signals that change if a hidden neuron weight is changed are shown in blue. The remaining signals are remembered in the signal table.

Table 2.1. Number of particular operations with and without a signal table required to calculate numerical gradient direction. Ni, Nh, No – number of input, hidden and output neurons.

type of operation	without signal table	with signal table
calculating sigmoid value	[No(Nh+1)+Nh(Ni+1)](No+Nh)	No(Nh+1)+Nh(Ni+1)(1+No)
adding incoming signals	$[Nh(Ni+1)+No(Nh+1)]^2$	2[No(Nh+1)+Nh(Ni+1)(1+No)]
calculating network error	[Nh(Ni+1)+No(Nh+1)]No	[Nh(Ni+1)+(Nh+1)]No

Table 2.2. Number of particular operations with and without a signal table required to calculate numerical gradient direction for the network structure 125-8-2 (Ni=125, Nh=8, No=2).

type of operation	without signal table	with signal table
calculating sigmoid value	10260 (100%)	3043 (29.7%)
adding incoming signals	1052676 (100%)	6084 (0.0058%)
calculating network error	2052 (100%)	2034 (99.1%)
total calculation time (experimental measurement)	100%	2.63%

The values in table 2.1 and 2.2 are given for a single training vector. If the gradient is determined on more vectors at once, the values must be multiplied by the number of vectors.

2.3.3. Analytically and Numerically Determined Gradient Directions

An interesting comparison can be made between the gradient direction determined analytically by BP (the same direction is used by all algorithms that use the backpropagation mechanism to calculate gradients) and the gradient direction determined numerically (given by the formula 2.31) [Kordos 2004d, 2005]. To obtain a good estimation of the gradient direction in a given point, *dw* must sufficiently small. As the experiments showed any *dw*<0.02 gives practically the same gradient direction. Thus the numerical gradient directions for *dw*=0.02 and *dw*=0.0002 do not differ noticeably. The plots in Fig. 2.6 are made for the normalized length of the gradient vector = 1. That is justified, since only the proportions between particular gradient components are meaningful and not their absolute values.

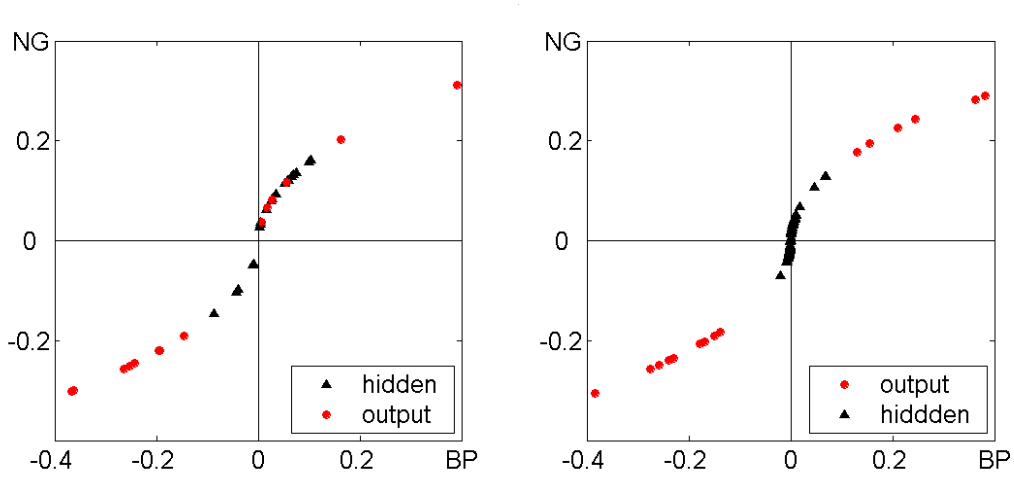

Fig. 2.6. Comparison of numerically (NG) and analytically (BP) determined gradient components in particular weight directions in the first training cycle. Left: Iris (4-4-3). Right: Thyroid (21-4-3).

The main difference between numerically and analytically determined gradient directions is that backpropagation interprets small gradient components (frequently hidden neuron weight components at the beginning of the training) as still smaller and big ones as still bigger. The differences between particular numerical and analytical gradient components are stronger for bigger networks and more complex datasets (for example the differences are stronger for the Thyroid than for Iris dataset, as shown in Fig. 2.6).

There are two reasons to assume that the direction towards minimum is closer to the numerical gradient than to the analytical one. First, in NG the gradient is determined directly and not assessed by backpropagation or by any other mechanism. Second, BP frequently does not converge. The effect is known as falling in spurious local minima. Spurious means that the minima are in the backpropagation-estimated gradient direction, but there exists a direction in which the trajectory can still go downward. In the experiments the networks were trained with BP. When they got stuck in a "local minimum", then from the same point in the weight space the trainings continued with NG in some cases were able to leave the apparent minimum and finally converge.

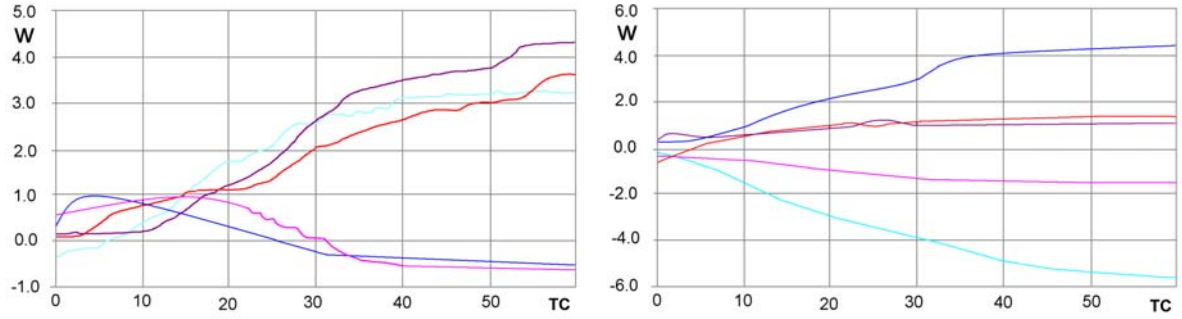

Fig. 2.7. Iris (4-4-3) trained with NG. Left: the first hidden neuron weight changes. Right: the first output neuron weight changes.

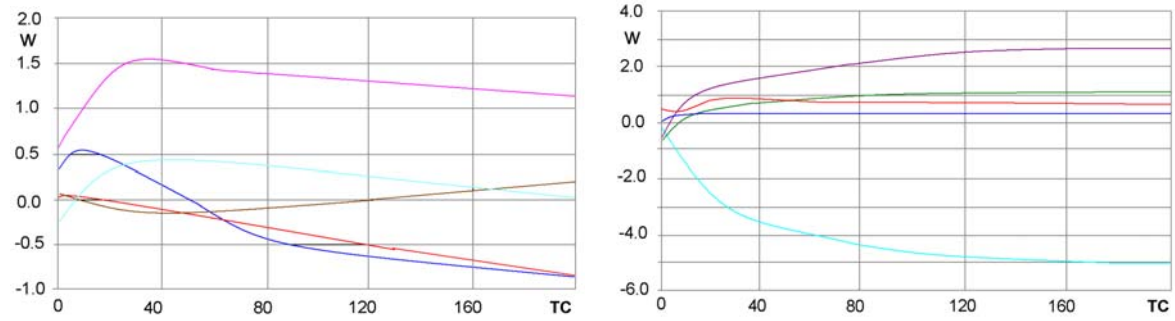

Fig. 2.8. Iris (4-4-3) trained with BP. Left: the first hidden neuron weight changes. Right: the first output neuron weight changes. The training started from the same initial weights as the NG training shown in Fig .2.7.

Figs. 2.7-2.8 present the results of experiments conducted with the network (4-4-3) trained on the Iris data with NG and with BP, starting from the same initial weights. The weights values during the training are shown for the first hidden and first output neuron. The first difference that can be noticed is that after the network is trained, the hidden weights are only slightly smaller than the output ones for NG training, while for BP training they are

significantly smaller than output weights. This could be expected, because of different gradient component distributions (Fig. 2.6). Thus, it can be concluded that the hidden layer weight values are underestimated in backpropagation-based trainings. This problem will be further discussed in chapter 2.4.3.

The second difference is that in BP the weights after some cycles grow very slowly, almost asymptotically. In NG the weight growth also slows down, but not so dramatically. The third difference is that some weights in NG and BP trainings after initially moving in the same direction, finally went in opposite directions and both trainings ended in different points of the weight space, although the initial starting point was identical.

2.3.4. Continuous and Discrete Search Space

The discrete NG is an algorithm, which assesses very roughly both the gradient direction and the optimal step length along this direction (Fig. 2.9-left). It works well for simple datasets, however in more difficult cases the continuous version of NG may be required. A comparison between discrete and continuous NG is presented in table 2.3.

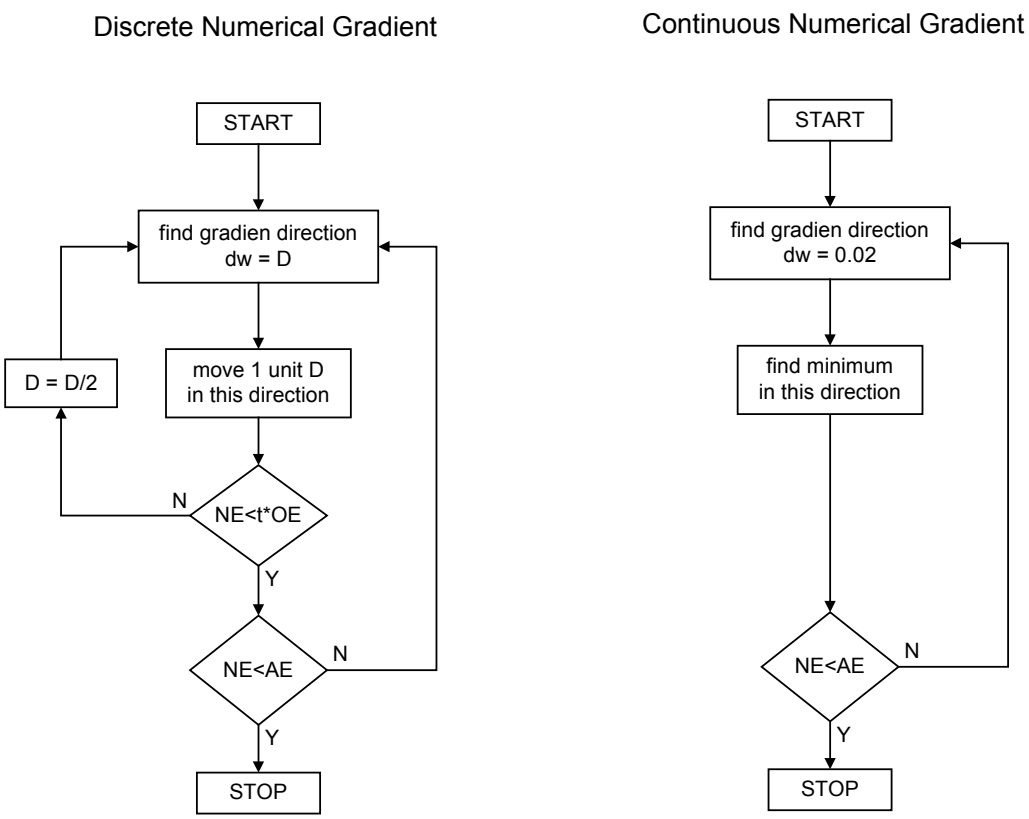

Fig. 2.9. Discrete and continuous numerical gradient algorithms. Any combination of these models is possible.

At the beginning of the discrete NG training both dw=0.5 and the precision of finding minimum D=0.5. If the error in the next training cycle is greater than 0.999 of the error in the previous training cycle, than both dw and D are divided by two (Fig. 2.9-left). The approximated gradient component $dE(w)$ can take only 3 values: –1 if the error increases after perturbing weight w with dw, 0 if it does not change and +1 if the error decreases. Other symbols used in Fig. 2.9: NE is the new error, OE – the (old) error in the previous training cycle, t is a threshold (t=0.995÷0.999), reducing the error to *AE (acceptable error)* terminates the training.

Numerical gradient in the discrete search space can be realized using only as few bits as four or five to represent all the weights and the value of the transfer function, which also can be discretized. Higher precision is required only to store the error value. The algorithm is simpler than NG in the continuous space, but it requires more training cycles to converge. Moreover, it is frequently unable to converge to such a good solution as the continuous version.

Both search spaces continuous and discrete can be realized with sigmoid, staircase and many other transfer functions [Duch 1999b]. The thesis concentrates on NG in continuous search spaces, and by "NG", the NG in the continuous search space will be understood.

Table 2.3. Comparison of discrete and continuous NG - number of training cycles required to achieve a given 10-fold crossvalidation accuracy (%test).

dataset	% test	network structure	discrete NG		continuous NG (2.37)	
			number of training cycles	total computational effort (scaled training time)	number of training cycles	total computational effort (scaled training time)
Iris	96	4-4-3	60	668	11	175
Breast	96	10-4-2	9	195	4	112
Mushrooms	98	125-4-2	82	3551	21	1070

The total computational cost of NG training consists of two terms: the cost of finding the gradient direction and the cost of finding the minimum along this direction. The higher the precision of finding the minimum along the gradient direction is the fewer training cycles are required to train the network, but the cost of finding the minimum grows.

While finding the gradient direction, only one weight is changed at a time and the signal table is used, thus the signals are propagated only through small fragments of the network. While finding the minimum along gradient direction, all the weights are changed at once and the signal table cannot be used, thus the signals must be propagated through the entire network. The ratio of the cost of finding the gradient direction c_{dir} to the cost of checking the error in one point along this direction c_{min} depends on the network structure. For networks with 50÷1000 weights usually c_{dir}/c_{min}=15÷30. Detailed explanation of how c_{dir}/c_{min} was calculated can be found in chapter 2.3.2, where the signal table is discussed.

There exists an optimal precision of finding the minimum along the gradient direction that allows for achieving a minimal cost of the training. As the experiments showed, using single or double parabolic approximation (the error is calculated 3 or 6 times) during the line search is frequently close to the optimal solution.

2.3.5. Gradient Direction and Optimal Next Step Direction

It would not necessarily be optimal to search for the minimum along the gradient direction (chapter 1.2.11). The statistically optimal search direction component *dS(w)* is a function of three variables: the network layer, the training cycle *Tc* and the gradient component *dE(w)*:

$$dS(w) = f(Tc, layer(w), dE(w)) \tag{2.32}$$

The function *f* can be considered as a product of two functions $f_{TL}=f(Tc,layer(w))$ that depends on the training phase and on the network layer and $f_D=f(dE(w))$ that depends on the gradient component in weight *w* direction.

$$dS(w) = f_{TL} f_D \tag{2.33}$$

The aim of the following reasoning is to determine how to select the functions f_{TL} and f_D to obtain the best approximation of the search direction *dS(w)* for a wide range of training datasets.

The error surface changes slower in the areas located further from its center. However, mostly output layer weights contribute to slower changes (Fig. 1.29-1.30). The differences between error surface sections in hidden weights directions at the beginning and at the final stage of the training are not so significant.

We are in a given point of the weight space and we want to assess the relation between the gradient component *dE(w)* in the direction of the weight *w* and the distance *mw* from the actual point to the error minimum in the direction *w* (Fig. 2.10). Both values *dE(w)* and *mw* can be obtained from the plots in Figs. 1.29-1.30. Some algorithms (wrongly) assume that *mw=f(dE(w))* is a linear correlation. However, it is clearly visible that at the beginning of the training the values *dE(w)* are greater in the output layer than in the hidden layer, while *mw* is smaller. Moreover, as the training progresses – the proportions change.

Although the function *mw=f(dE(w))* cannot be a priori defined for any particular weight, some statistical correlations are quite easy to observe. Thus *mw=f(dE(w))* should be rather thought of as a statistical distribution than as a function given by an analytical formula. As many statistical distributions must be maintained during the training as the number of neuron layers with optimized weights: one for each hidden layer and one for the output layer. The distributions must be gradually modified as training progresses, since the error surface landscape changes.

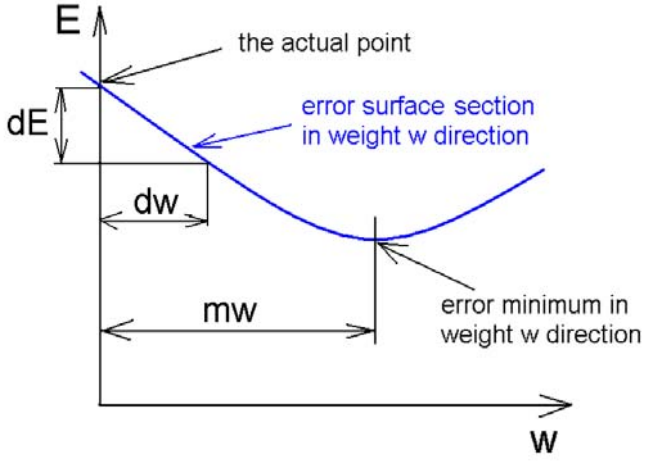

Fig. 2.10. *dE(w)* is proportional to the error derivative in the actual point with respect to weight *w*. Since *dE(w)* is not proportional to *mw*, we search for the value *dS(w)* that allows for a better approximation of *mw*.

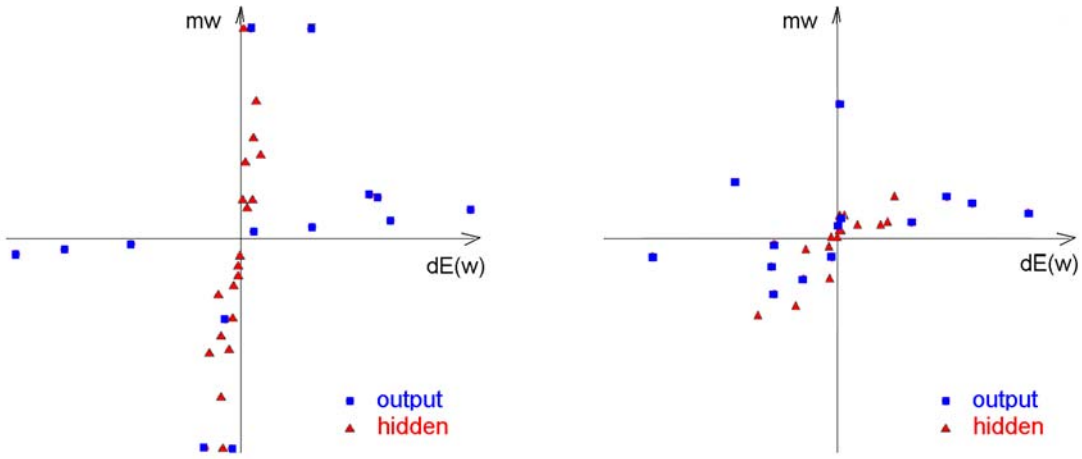

Fig. 2.11. Dependence between the gradient component *dE(w)* and the distance from the actual point to the error minimum *mw* in a given weight direction at the beginning (left) and at the end of the training (right) for Iris (4-4-3).

In the first approximation we can assume a linear dependence between the optimal search direction component *dS(w)* and the gradient component *dE(w)* within the same training cycle and the same network layer:

$$dS(w) = f(Tc, layer(w)) \cdot dE(w) = f_{TL} \cdot dE(w) \tag{2.34}$$

f_{TL} is the more important factor and using only f_{TL} we can get a better approximation of the direction toward the minimum than using only f_D. f_{TL} equals 1 for the output layer and for hidden layers it can either equal 1 or decrease gradually from a higher value at the initial phase of the training down to 1 at the final stage of the training.

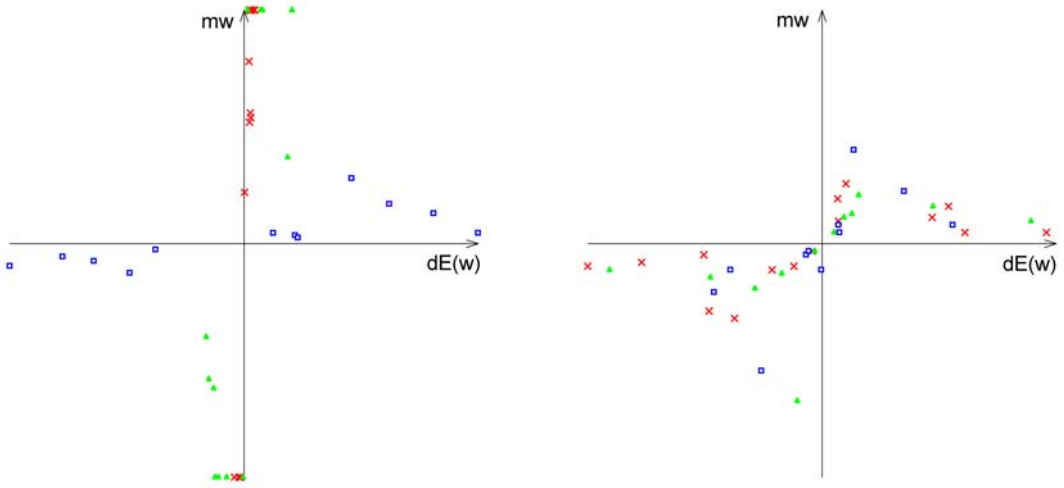

Fig. 2.12. Dependence between the gradient component *dE(w)* and the distance from the actual point to the error minimum in a given weight direction *mw* at the beginning (left) and at the end of the training (right) for Iris (4-3-3-3). Red cross = first hidden (counting from input), green triangle = second hidden, blue square = output layer.

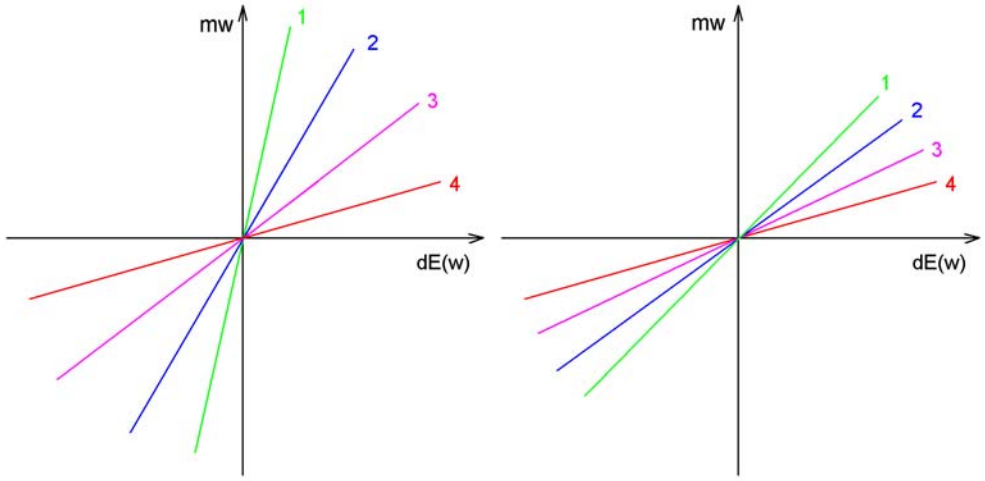

Fig. 2.13. Linear approximation of the dependence between *mw* and *dE(w)* for hidden layers; 1 – training cycle 1 through 3; 2 – training cycles 4 through 6; 3 – training cycles 7 through 11; 4 – training cycles above 11. The red line (4) also approximates the dependence between *mw* and *dE(w)* for the output layer in any training cycle. Left: the hidden layer in three-layer networks and the first hidden layer in four-layer networks. Right: the second hidden layer in four-layer networks.

In case of a four-layer network, the dependence between *mw* and *dE(w)* for the second hidden layer can be approximated with a line situated between the lines approximating the dependencies for the first hidden and for the output layer, however at the end of the training all the three lines converge to one.

Thus, the formula (2.34) can be written as:

$$dS(w) = (1 + a \cdot \exp(-b \cdot Tc)) \cdot dE(w) \tag{2.35}$$

where the typical, experimentally determined, optimal values are:

for 3-layer networks:
a=0 for the output layer
a=10÷20 for the hidden layer
b=0.10÷0.15

for 4-layer networks:
a=0 for the output layer
a=3÷5 for the second hidden layer (between the first hidden and the output layer)
a=10÷20 for the first hidden layer (between the input and the second hidden layer)
b=0.10÷0.15.

This modification works significantly better, but as experiments showed, too big *dE(w)* do not correspond to *mw* linearly. Thus, they should be limited to a certain value.

There are several ways to do it. For example, instead of a linear function, a linear function with a constant value outside a certain range (a saturated linear function) can be used:

$$\begin{aligned} dS(w) &= f_{TL} \cdot dE(w) \quad \text{for } -dE_0 \le dE(w) \le dE_0 \\ dS(w) &= f_{TL} \cdot sign(dE_0) \cdot dE_0 \quad \text{otherwise} \end{aligned} \tag{2.36}$$

where $f_{TL} = 1 + a \cdot \exp(-b \cdot Tc)$.
Another possibility is to use a square root of *dE(w)*

$$\begin{aligned} dS(w) &= f_{TL} \cdot sign(dE(w)) \cdot \sqrt{|dE(w)|} \quad \text{for } -dE_1 \le dE(w) \le dE_1 \\ dS(w) &= f_{TL} \cdot sign(dE_1) \cdot \sqrt{|dE_1|} \quad \text{otherwise} \end{aligned} \tag{2.37}$$

where $f_{TL} = 1 + a \cdot \exp(-b \cdot Tc)$.
Still another option is to use a non-monotone transfer function, for example:

$$dS(w) = f_{TL} \cdot dE_2 \cdot sign(dE(w)) \cdot dE^2(w) \cdot \exp(-dE_2 \cdot dE^2(w)) \tag{2.38}$$

where $f_{TL} = 1 + a \cdot \exp(-b \cdot Tc)$, dE_0, dE_1, dE_2 are proportional to the standard deviation σ of gradient components in a given training cycle ($dE_0 = 4\sigma$, $dE_1 = 8\sigma$, $dE_2 = 2\sigma$).

A series of experiments was conducted to assess which approximation would be the best. Instead of using the least square error as an index of the approximation quality, the network convergence was observed. The experimental results are presented in Table 2.4.

Though the differences are not big, on average the formula (2.37) gives the best performance and it will be used further. This formula was tested with various exponents from (0;1), not only with 0.5. However, the exponent 0.5 seems to be the most optimal one. It is interesting that the convergence speed for the exponent being zero and being one are very

similar. RPROP is an algorithm, which takes into consideration only the sign of the derivative, but not its value (exponent=0) and it performs not worse than BP (exponent=1). Sin-Chung Ng [Ng 2004] has also recently proposed that the gradient components calculated by backpropagation should be taken in power 0.5, however their main reason for that is increasing the small gradients to accelerate the training in the flat error surface areas.

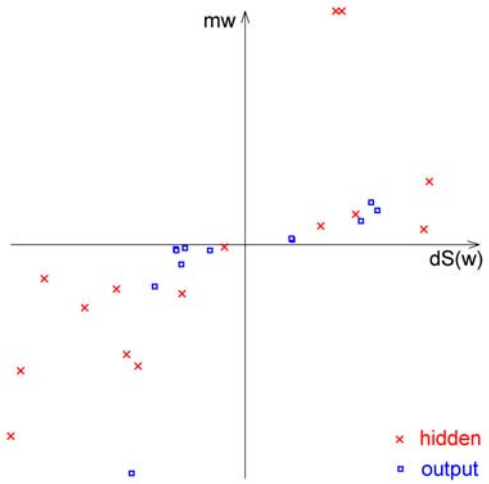

Fig. 2.14. Iris 4-3-3. Dependence between the search component *dS(w)* and the distance *mw* from the actual point to the error minimum in a given weight direction at the beginning of the training for Iris (4-4-3) calculated with (2.37), which displays the best convergence properties of the methods considered here.

Table 2.4. Average number of training cycles required to reach a given accuracy on the training set with various versions of NG.

dataset	Iris		Ionosphere		Thyroid	
network	4-4-3		43-4-2		21-4-3	
accuracy	92%	98%	90%	96%	94%	97%
gradient (2.33)	18	30	20	70	-	-
Tc optimized (2.34)	12	18	12	64	32	-
linear + limit (2.36)	9.3	14	12	42	28	52
sqrt + limit (2.37)	8.0	11	12	42	18	40
a·dE·dE·exp(-a·dE) (2.38)	9.5	14	20	44	-	-

2.3.6. Error Surface Curvature and Second Derivative

The second order analytical gradient based MLP training algorithms, such as Levenberg-Marquardt (LM) use not only the information contained in the first but also in the second derivative (error surface curvature). They assume that the optimal search component in the direction *w* is approximately proportional to the ratio of the first to the second derivative. This assumption seems to be right, because LM algorithm displays much better convergence properties than first order methods, but on the other hand its memory requirements and calculation times grow rapidly with the network and dataset size, which causes that in practice LM can be used only for small networks and small datasets (chapter 2.1.1.6). However, the second order methods are not very stable. LM sometimes finds a very good solution but frequently does not converge at all.

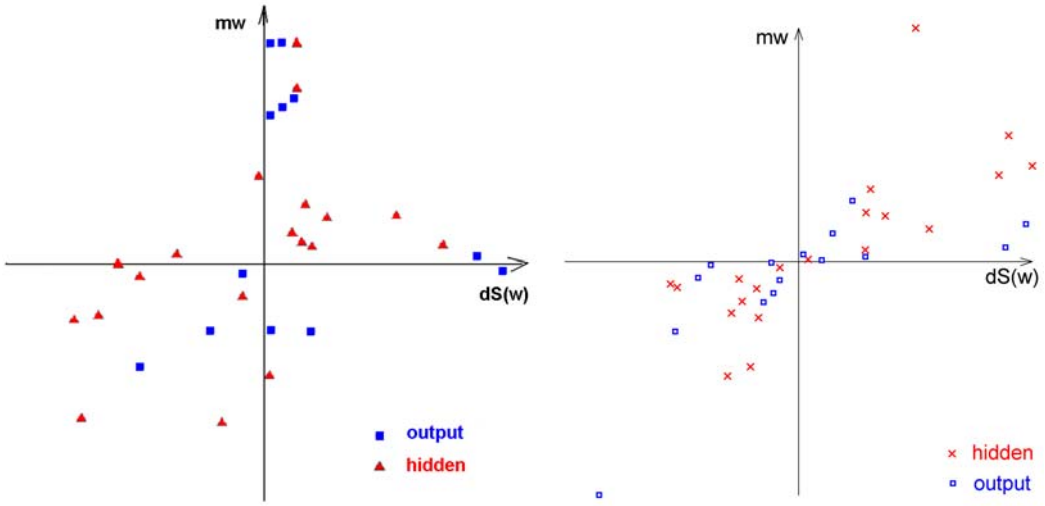

Fig. 2.15. Left: Dependence between the search component *dS(w)* and the distance from actual point to the error minimum in a given weight direction *mw* at the beginning of the training for iris (4-4-3) trained with: left – LM, right – NG using equation (2.40).

The equation (2.37) aims at achieving a better convergence than the first order methods, while being still suitable for large networks and large datasets. Nevertheless, it would be interesting to see the correlation between the ratio of the numerical gradient component to the curvature of the error surface section in a single weight direction and the distance to the error minimum in that direction. To achieve this, the numerical second derivative d^2w is calculated as the curvature measure, only in single weight directions. We already know the error *E(w)* in the actual point and the error in the point *E(w+dw)*. In order to assess the curvature, we must calculate the error in one additional point, for example in *(w+0.5dw)*:

$$d^2E(w) = 0.5 \cdot \big(E(w) + E(w + dw)\big) - E(w + 0.5 \cdot dw) \tag{2.39}$$

Thus calculating the second derivative requires twice as much calculation as calculating only the first derivative. Therefore, this approach may be justified only if it allows

for reducing the number of training cycles at least twice in comparison to the number achieved with (2.37).

$$dS(w) = \frac{dE(w)}{|d^2E(w)|} = \frac{E(w) - E(w+dw)}{|d^2E(w)|} \tag{2.40}$$

$d^2E(w)<0$ means that the error surface in weight w direction is concave. It is fortunately concave in the prevailing number of points covered by the training trajectory. That is obvious, because the trajectory tends to occupy rather the error surface ravines than ridges. The absolute value of $d^2E(w)$ is taken in (2.40), because the sign of $dS(w)$ is determined only by the direction of error decrease and not by the error surface being concave or convex.

Although using the second order information gives on average a slightly better linearity of the correlation between mw and $dS(w)$, it allows to train the network on average in the same number of training cycles as the search direction given by (2.37), as shown in Table 2.5. Moreover, sometimes problems with convergence may occur and the amount of calculations is doubled. Thus, it is not suggested to use this method.

The most efficient solution would be probably when we get a linear dependence $dS(w)=f(mw)$, except for the cases when a minimum in a given direction lies in infinity or very far - then the move in this direction must be limited. In this aim, the minimum in each weight direction must be searched for separately. Searching for the minimum in each weight direction separately will be computationally costly and as it is known from the experiments, the results expressed by the number of training cycles improve only a little. However, after some modifications this idea can work exceptionally well (chapter 2.4).

2.3.7. Numerical Gradient with Momentum

The idea of momentum is to accelerate the training convergence by using the information about the weight changes in the previous training cycle while determining the changes in the actual training cycle. Usefulness of this approach can be explained in two ways: either using the information about single weight changes or using the information about MLP error surface and learning trajectory shapes.

The average changes of a given weight in two successive training cycles are usually similar. Therefore, it seems reasonable to force greater changes in the same direction in order to minimize the required number of training cycles. However, forcing the algorithm to go beyond the minimum in the gradient direction does not work very well. After making such an oversized step we reach a point on the opposite slope of the error surface ravine. The gradient direction in that point strongly differs from the gradient direction close to the bottom of the ravine. Consequently, the trajectory will oscillate from one side of the ravine to the other. Thus, when the step size increases beyond the minimum in a given direction, also the direction must be corrected. This leads to a conclusion that error surface ravines create arcs. If the same distance along the arc must be covered in fewer steps, then it is obvious that the angle between the directions of the successive steps must be smaller. That can be obtained by using a weighted sum of the previous step and the line from the current position to the minimum in the gradient direction. This method known as momentum can be realized with

NG in a similar way as it is realized with BP. The following formula expresses the change of weight *w*, using NG with momentum:

$$dw(Tc) = mometum \cdot d(w_{Tc-1}) + d(w_{Tc}) \tag{2.41}$$

where *d(w)* is the distance that would be covered in the weight *w* direction if the trajectory moved in the gradient direction. With momentum the weights grow quicker, especially at the beginning of the training.

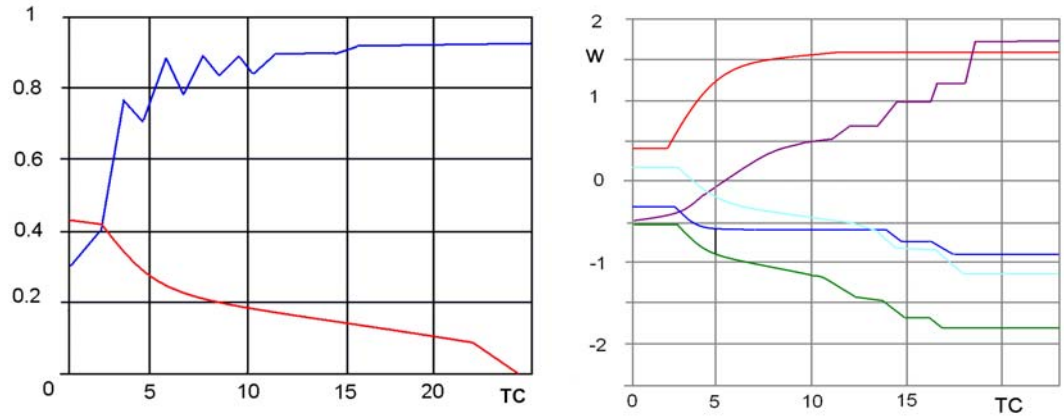

Fig. 2.16. Iris (4-4-3) trained with NG without momentum. Left: MSE (red) and classification accuracy (blue) on the training set. Right: values of the first hidden neuron weights.

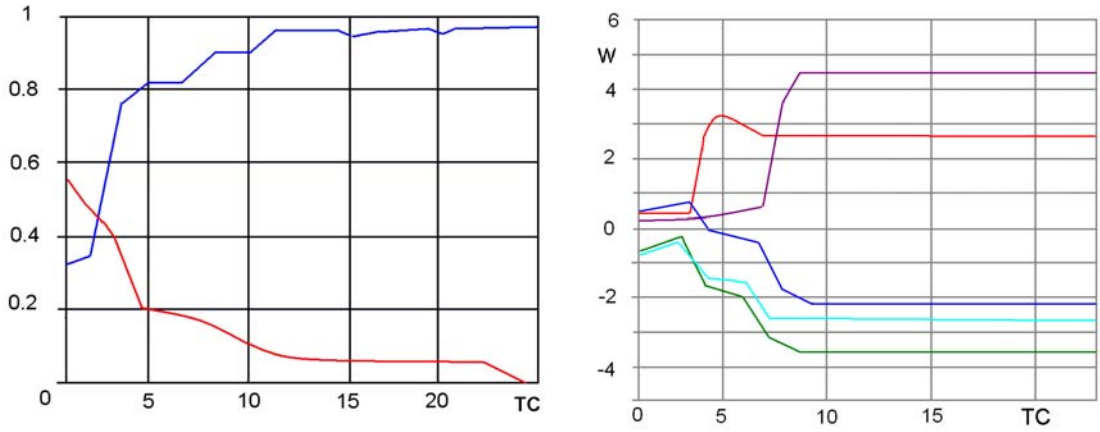

Fig. 2.17. Iris (4-4-3) trained with NG with momentum. Left: MSE (red) and classification accuracy (blue) on the training set. Right: values of the first hidden neuron weights.

Also another interesting effect caused by momentum can be observed with some datasets: many weights do not grow slowly to infinity or to very big values, but stabilize at constant values after some training cycles (Fig 2.17). The stabilization occurs already in the area of network convergence. A network trained with NG without momentum requires much more training cycles to reach such big weights values. Moreover, momentum decreases the oscillations of classification accuracy on the training set at the beginning of the training.

Momentum can be also used to minimize trajectory oscillations. The oscillations can appear if we use the discrete version of NG, where the minimum along the gradient direction is not localized, but a constant step is used. With continuous NG, the oscillations are limited to the accuracy of finding the minimum along the gradient direction and are too small to have practical influence on the training process. Thus, there is no need to reduce them by averaging the directions from several iterations.

Momentum works very well for the Iris dataset but it does not work equally well for every dataset. The optimal momentum value must be chosen individually for each dataset. Usually higher momentum values are possible for smaller networks. Moreover, while used with NG, it must be sometimes switched off at the final stage of the training, since after accelerating the initial stage of the training, the momentum term can prevent the network from the final convergence (the weight stabilization can occur too early).

Other possibilities of decreasing training times include weight pruning and freezing (see chapter 2.4.3), using border vectors (chapter 2.5.1) and updating the weights after only a part of the training set is propagated through the network (chapter 2.5.2).

2.3.8. Experimental Comparison of various NG Methods

Table 2.5. Average number of training cycles required to obtain a given accuracy on the training set with various versions of NG. The optimal momentum was determined experimentally for each dataset.

dataset	Iris		Ionosphere		Sonar	
network	4-3-3		34-4-2		60-8-2	
accuracy	90%	96%	90%	96%	90%	99%
gradient (2.31)	18	30	20	56	12	32
optimized (2.37)	7.9	11	12	32	10	30
second derivative (2.40)	8.1	11	12	-	17	45
momentum (2.41)	8.0	11	12	24	14	60
optimized+momentum	7.1	10	8.2	16	12	45
step to a minimum in each weight direction	5.1	8.6	5.4	13	6.8	25

Figs. 2.18-2.21 are presented as comments to Table 2.5.

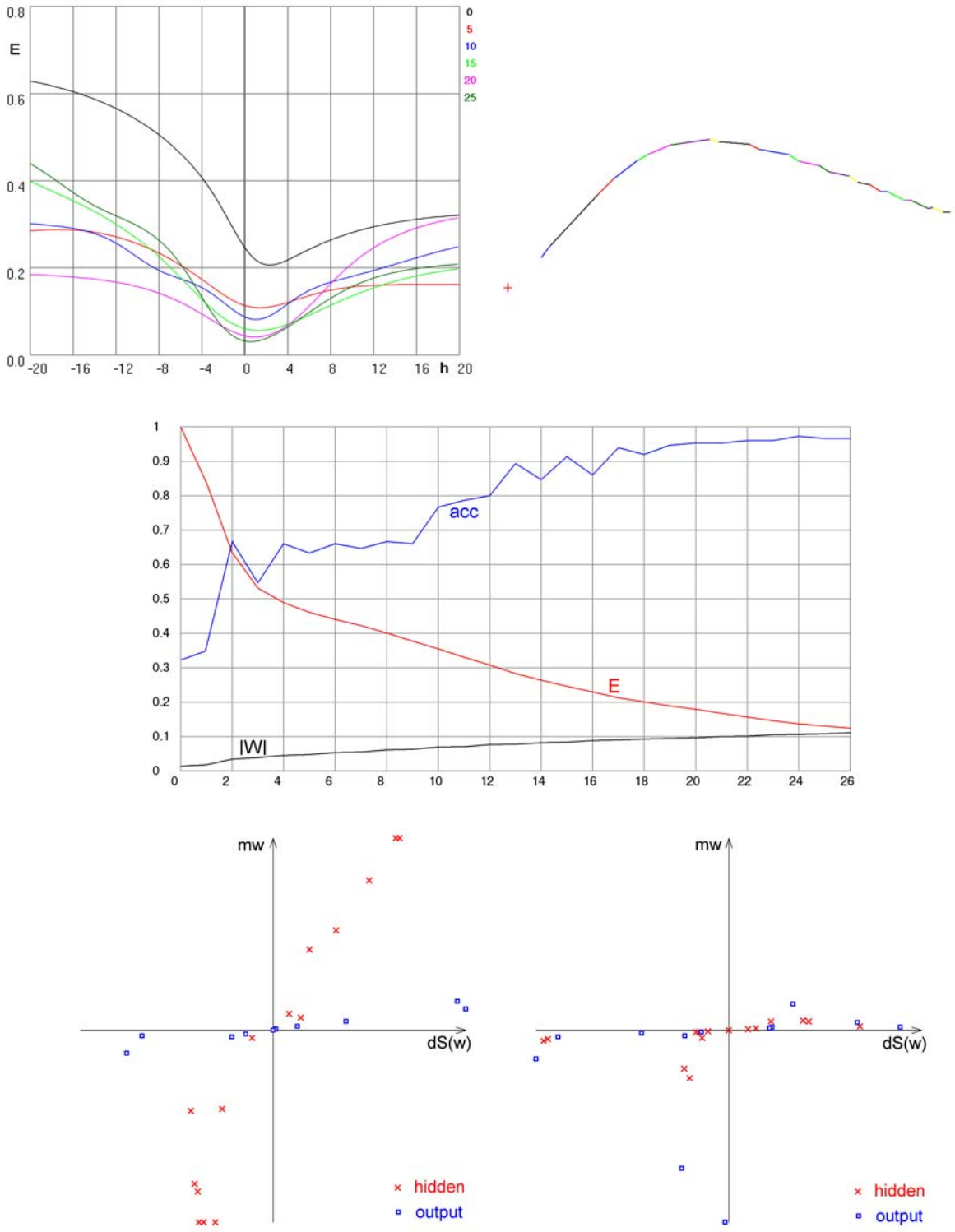

Fig. 2.18. Standard NG (2.13) for iris 4-3-3. Left-top: error surface sections in the search directions ***dS*** at the starting point and in training cycles: 5, 10, 15, 20 and 25. Right-top: PCA-based learning trajectory projection. Middle: on vertical axis: *E* - MSE error, *acc* - accuracy, |***W***| - length of weight vector, on horizontal axis: training cycle (the vertical axis is in relative values that can be compared among pictures 2.14-2.17). Left-bottom: Dependence between search components *dS(w)* and the distance *mw* from actual point to the minimum in weight *w* direction in the first training cycle. Right-bottom: *dS(w)* and *mw* in the 25th training cycle.

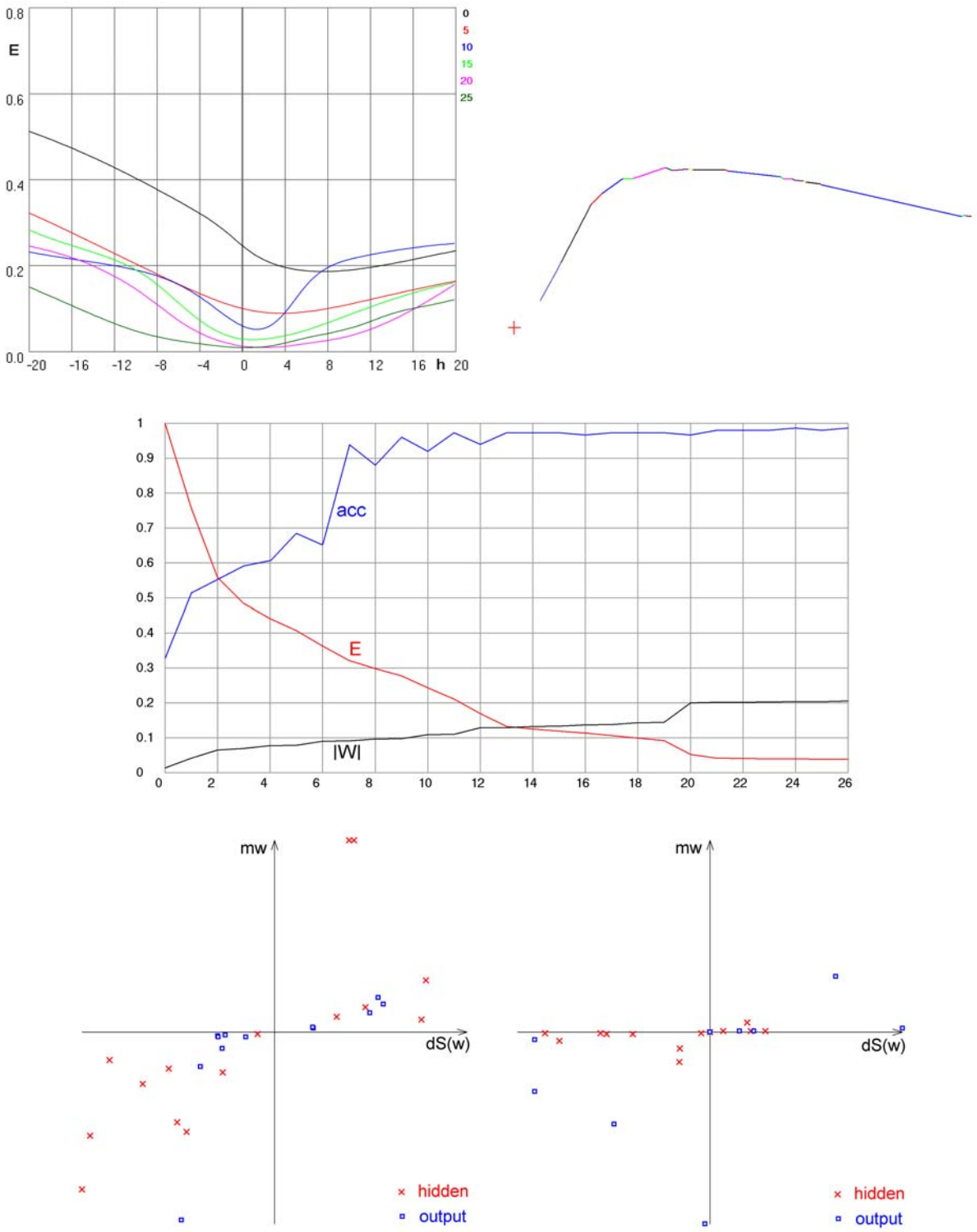

Fig. 2.19. NG with optimized direction (2.37) for iris 4-3-3. Left-top: error surface sections in the search directions ***dS*** at the starting point and in training cycles: 5, 10, 15, 20 and 25. Right-top: PCA-based learning trajectory projection. Middle: on vertical axis: *E* - MSE error, *acc* - accuracy, |***W***| - length of weight vector, on horizontal axis: training cycle. Left-bottom: Dependence between search components *dS(w)* and the distance *mw* from actual point to the minimum in weight *w* direction in the first training cycle. Right-bottom: *dS(w)* and *mw* in the 25th training cycle.

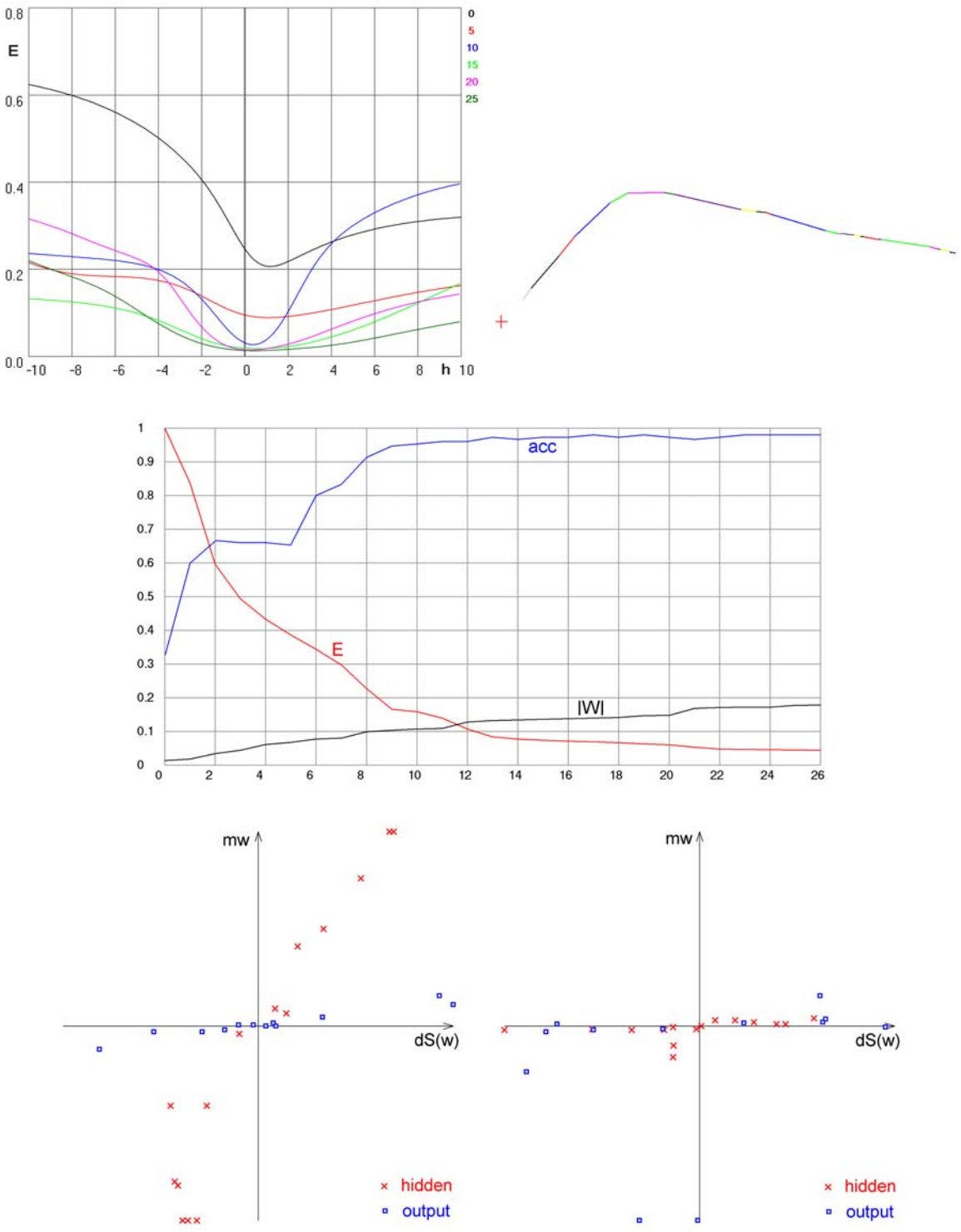

Fig. 2.20. NG with momentum = 0.25 (2.42) for iris 4-3-3. Left-top: error surface sections in the search directions ***dS*** at the starting point and in training cycles: 5, 10, 15, 20 and 25. Right-top: PCA-based learning trajectory projection. Middle: on vertical axis: *E* - MSE error, *acc* - accuracy, |***W***| - length of weight vector, on horizontal axis: training cycle. Left-bottom: Dependence between search components *dS(w)* and the distance *mw* from actual point to the minimum in weight *w* direction in the first training cycle. Right-bottom: *dS(w)* and *mw* in the 25th training cycle. Momentum does not work so well on every dataset..

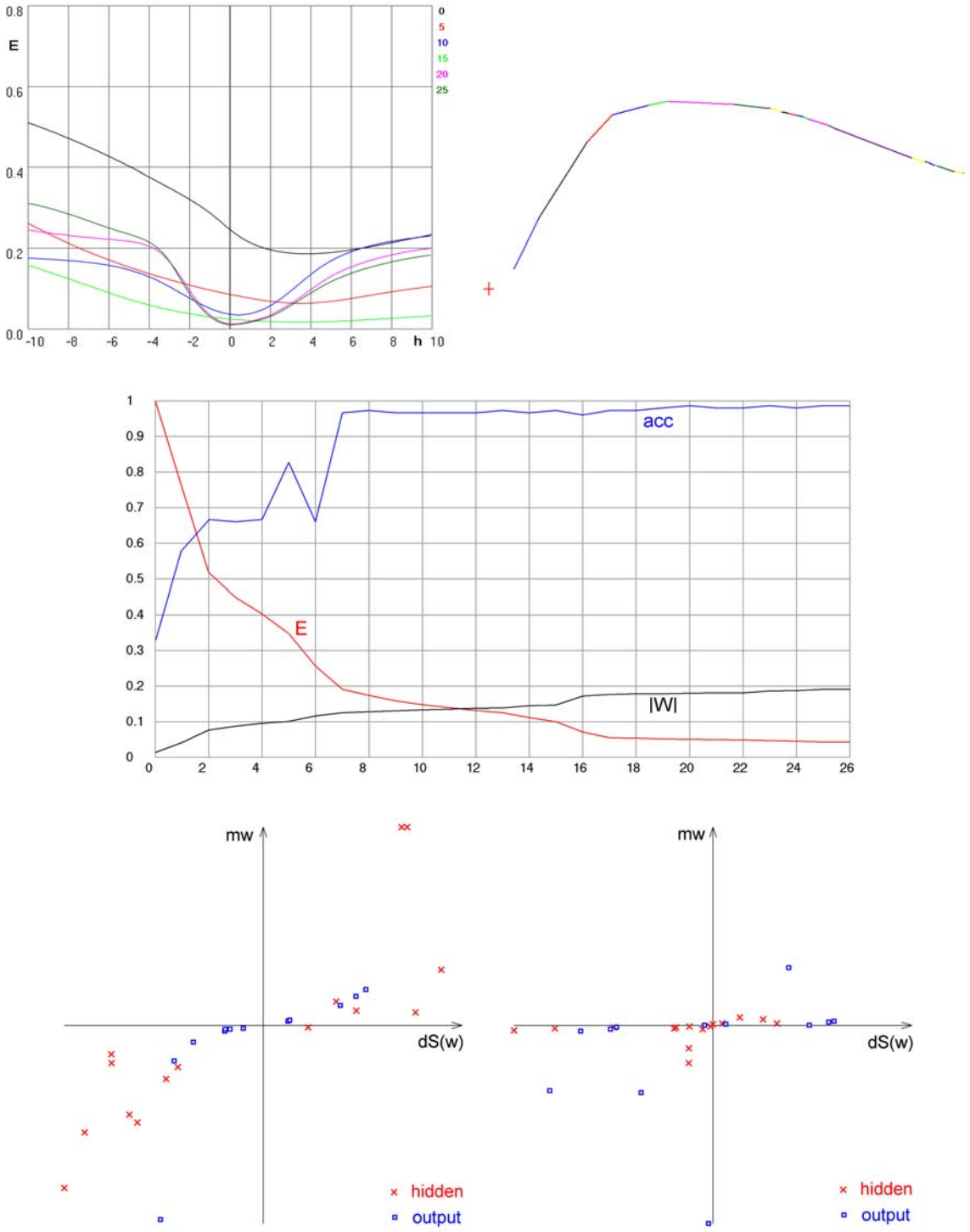

Fig. 2.21. NG with optimized direction and momentum (2.43) for iris 4-3-3.Left-top: error surface sections in the search directions ***dS*** at the starting point and in training cycles: 5, 10, 15, 20 and 25. Right-top: PCA-based learning trajectory projection. Middle: on vertical axis: *E* - MSE error, *acc* - accuracy, |***W***| - length of weight vector, on horizontal axis: training cycle. Left-bottom: Dependence between search components *dS(w)* and the distance *mw* from actual point to the minimum in weight *w* direction in the first training cycle. Right-bottom: *dS(w)* and *mw* in the 25th training cycle.

2.3.9. Conclusions

The numerically and analytically determined gradient directions in MLP weight space differ. Though the difference is usually not great, its cumulative effect during the training can cause that the algorithms based on numerical and analytical gradients will find quite different solutions while starting from the same initial weights. A significant difference exists between any of the gradient directions and the optimal next step direction. The common tendency of many training algorithms based on analytical gradient is to underestimate the modifications of the hidden layer weights (Fig. 2.7, 2.8 and 2.28).

The discrete NG is the simplest version of the numerical gradient algorithm. It has lower computational cost per one training cycle, however it requires more training cycles and its total computational effort is higher than that of continuous NG. There exists an optimal precision of finding the minimum along the gradient direction or along the modified search direction, which allows for the lowest training costs.

The optimized direction ***dS*** allows for longer steps and thus the training can be done in fewer steps. Since this does not impose any additional overhead, it is advocated to use the optimized direction ***dS***. Except for the optimized direction ***dS***, each other enhancement increases the training speed by reducing the amount of information calculated by NG algorithm (including semi-batch or on-line training or using border vectors – see chapter 2.5.1–2.5.2). The information cannot be reduced too much, since then the training will not be able to converge. For that reason if some of the methods are combined together, each of them should modify the basic training algorithm less than if used separately (for example the optimal momentum can be 0.4 with the batch training and 0.2 with the semi-batch training).

2.4. Variable Step Search Algorithm

2.4.1. In-Place versus Progressive Search

In the in-place search used by NG, all the weight changes are examined in the same point on the error surface and then a single step is made in the calculated direction. In the progressive search, after each weight change is examined, immediately a step in this weight direction is made and the next weight changes are examined already in the new point on the error surface.

As the experimental results showed (chapter 2.3.8), the best method of searching for the next step direction is to find a minimum in each weight direction separately (using any line search method) and then to move to that point.

However, if the minimum in the weight w_1 direction is found, we can immediately move to that minimum and then search for the minimum in the weight w_2 direction being already in the new point. Then we move to the minimum found in the weight w_2 direction and so on. Always a step in a given weight direction is made immediately after the minimum in that direction is found, while all remaining weights are not changed. Thus, there are as many steps in orthogonal directions during one training cycle as the number of weights. Many experiments aiming at determining the optimal weight change sequence were performed, however the various sequences did not have significant influence on the training efficiency. It cannot be concluded that any sequence produces the same results because it is also possible that the optimal sequence has not been found so far. Therefore the weights are changed one be one, first all weights from the hidden layer than all weights from the output layer, or first all weights from the output layer and then from the hidden layer. Only after detecting that changing a given weight does not change the error, the weight is frozen or pruned (chapter 2.4.3).

The computational cost per training cycle is the same as for the NG in-place search, but as experiments showed, several times fewer training cycles are required to train the network. Moreover, the progressive search method is usually able to find better solutions than the in-place search. Frequently the quality of the solution is the most important factor and the training time is less important or not important at all, especially for small datasets.

The progressive search as an MLP training algorithm is more stable and allows for training the network in a fewer training cycles than any other method considered so far. However, several modifications are still required to decrease the computational cost of the solution.

There are at least three methods of minimizing the cost. The first method is remembering neuron signals in the signal table instead of calculating them every time (chapter 2.3.2). Signal tables can reduce the cost up to hundreds times. The second method is to use appropriate search heuristics while determining the weight values (chapter 2.4.2). The cost reduction due to the heuristics is difficult to assess precisely, because it depends on many factors. After applying these two methods, this algorithm performed exceptionally well and the name *variable step search algorithm* (VSS) was proposed for it [Kordos 2004b].

The third, optional, method is to use a staircase transfer function instead of sigmoids, which can be used since no gradient and no derivatives are calculated by VSS. The values of the staircase functions can be read from arrays instead of being calculated each time. The time of calculating the neuron signals is ranks of order shorter, but the total training times are about 3÷30% shorter compared to logistic sigmoids and 4÷40% shorter compared to hyperbolic tangents, depending on the network structure and the training algorithm.

2.4.2. Determining Weight Values

The simplest search-based algorithm works in the following way: in one training cycle the value of *dw* is added to or subtracted from a single weight *w*. If the error decreases then the change is kept, otherwise it is rejected. Then *dw* is added to or subtracted from the next weight and again the error is calculated, until the changes of all weights are examined. *dw* can be gradually decreased each training cycle. This algorithm used for logical rule extraction from MLP networks with not fully connected layers will be presented in chapter 3.2.

VSS is the modified version of the simplest search-based algorithm, in which *dw* is not constant, but dynamically adjusted independently for each weight during a rough minimization in each weight direction. VSS was designed taking the advantage of MLP error surface properties that its steepness in different directions varies ranks of orders, and the ravines in which the MLP learning trajectories lay are usually curves, slowly changing their directions [Kordos 2004a, 2004c][Gallagher 2000, 2003]. Basing on the properties we can expect that an optimal *dw* for the same weight in two successive training cycles will not differ much while *dw* for different weights in the same training cycle may differ ranks of order.

In each training cycle *i* the first guess of *dw(w,i)* for a given weight *w* might be the value *dw(w,i*−1) that the weight changed about in the previous training cycle. However the detailed experimental analysis of the algorithm behavior leads to the conclusion that for most cases the least number of calculations is obtained when the first guess is *dw(w,i)*= *c1*·*dw(w,i*−1), with *c1* in the range *0.3÷0.4*, in spite that statistically the ratio of *dw(w,i)/dw(w,i*−1) is close to 1.

Fig. 2.22. shows a diagram for determining *dw* of a single weight in one training cycle. Before the training starts, the weights are initialized with random values from the interval (−1;+1). Initializing all hidden layer weights with zero values and setting the first guess *d0* of each weight change to a large value is an effective method of feature reduction in the first training cycle. The larger *d0* (0.5, 1, 2) is the more features are eliminated from further training. After the first training cycle all hidden weights that still equal zero are pruned and *d0* is again set to a smaller value.

In the first training cycle *d=d0=0.2÷0.3*. Since *dw(w,0)=0,* for each weight *w* in the first training cycle the first guess is *dw(w,1)=d0*. The ravine on the error surface is narrow close to the algorithm starting point. Thus setting *d0>0.5* frequently causes that the trajectory cannot well fit into the ravine bottom and some weights oscillate while others do not change at all during some initial training cycles, resulting in a slow training or convergence problems.

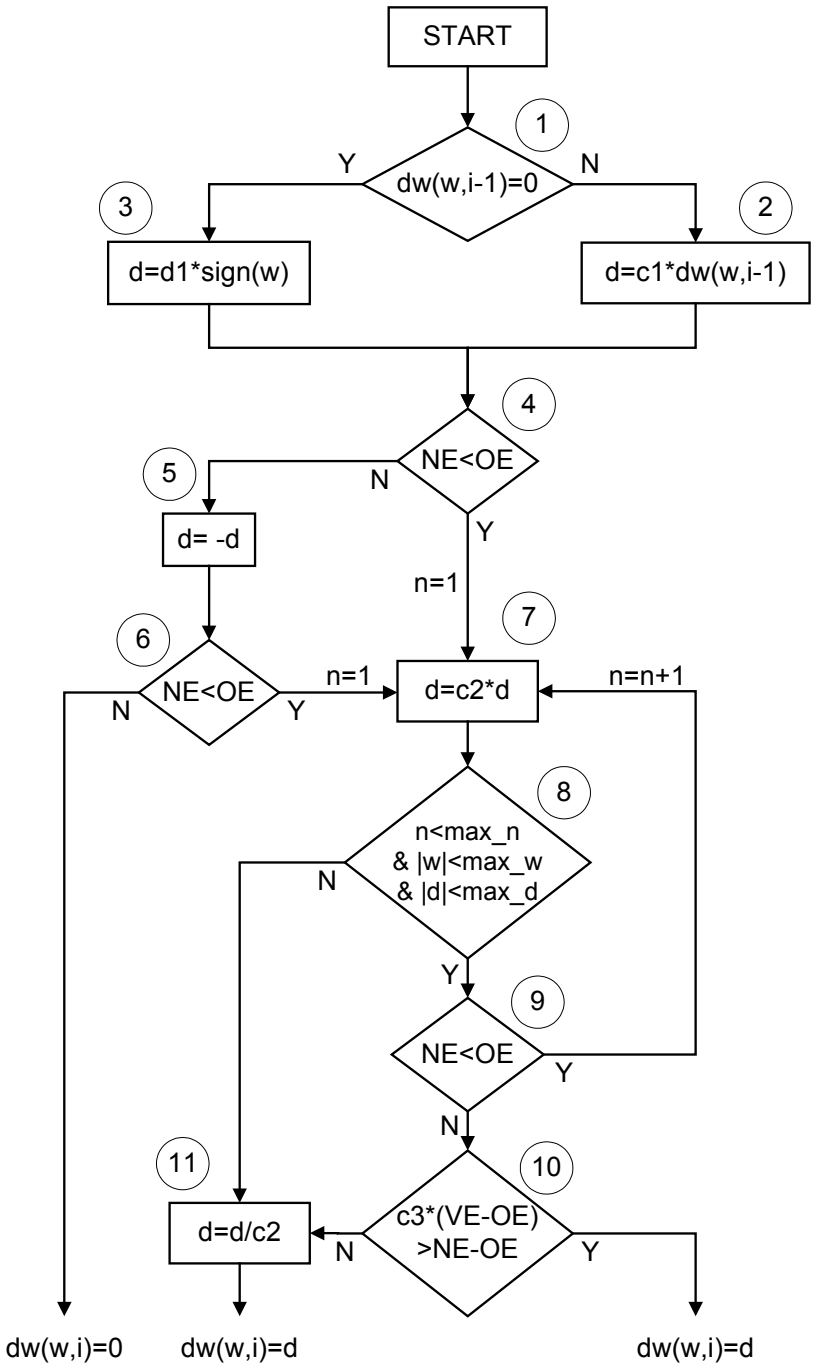

Fig. 2.22. Determining a single weight value in one training cycle of the Variable Step Search Algorithm. Typical values of constants c1, c2, c3, d1 are given in Table 2.6.

Comments to the VSS algorithm diagram (Fig. 2.22):

1: if *dw(w,i)=0* then goto 3.
2: the value *c1·dw(w,i-1)* is added to the weight *w*.
3: if the weight *w* did not change in the previous training cycle try to add (or subtract) to it a smaller value *d=d1·sign(w)*. Each weight is more likely to change in the same direction in the next training cycle. For that reason *d1* is multiplied by *sign(w)*

to minimize the number of operations. If the situation repeats twice or more than the weight can be optionally frozen.

4: If the new error *NE* after the change is smaller than the old error *OE* before the change then the direction of the change is correct, goto 7.

5: otherwise change the direction of search $d=-d$.

6: If the new error *NE* after the change is not smaller than the old error *OE* before the change then do not change the weight.

7: Search for an approximate minimum along this direction; set $d=c2 \cdot d$

8: If $n<max\_n$ and $|w|<max\_w$ and $|d|<max\_d$ then goto 9 else goto 11. *max_n* is given to prevent the loop through points 7-9 from being executed too many times. Maximal acceptable values for a single weight *max_w* and for a single weight change *max_d* provide an optional way of weight regularization and can be set to infinity if weight regularization is not required or already provided by a standard penalty term added to the error function.

9: If the new error *NE* after the change is smaller the than old error *OE* before the change then goto 7 else goto 10.

10: If $c3 \cdot (VE-OE) > NE-OE$ then accept that point in spite that the error in the previous point was a bit lower else return to the previous point (goto 11). *VE* is the last error before *OE*, i.e. *NE=error(n)*, *OE=error(n-1)*, *VE=error(n-2)*. It works like a momentum with standard backpropagation and is likely to bring gain in the next training cycle.

11: $d=d/c2$. Return to the previous point.

Table 2.6. VSS parameters with sigmoid slope=1. The sensitivity column contains the range of a parameter within which the VSS effectiveness is at least 90% of that for the optimal parameter. The values are only approximate and do not include interactions between parameters.

parameter	optimal value	sensitivity (10% range)	explained in point No.
d0	0.2	0.10÷0.30	above Fig.2.18.
d1	0.03	0.01÷0.10	3
c1	0.33	0.22÷0.44	2
c2	2.0	1.5÷3.3	7
max_n	4	3÷8	8
c3	0.3	0.1÷0.5	10

Many experiments with various weight updates strategies were made. On average the error is calculated about 3 times while determining a single weight value in one training cycle. It is possible to reduce this number but this leads to a higher number of training cycles. It is likely that a more efficient weight update scheme exists, however it has not been found so far.

2.4.3. Analysis of Weight Changes

The VSS algorithm is very convenient for visualization purposes since it changes only one weight at a time, which allows us to assess the influence of single weights on the convergence process.

The plots presenting error value as a function of epoch number are widely used in literature. From Fig. 2.23 it can be seen that the weight changes (absolute values) in the first training cycle are either zero or the initial change d0. As the training progresses some weights change slower and some faster. After several training cycles it is clearly visible which weights do not change any more or their little changes do not significantly influence the error value and these weights can be frozen or pruned.

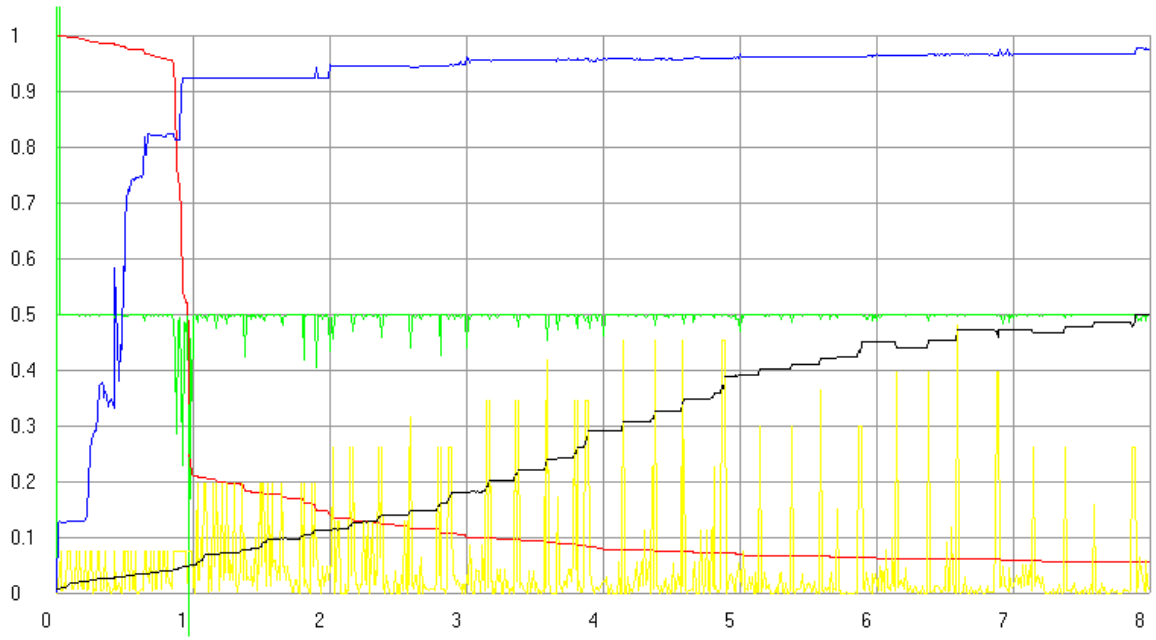

Fig. 2.23. Thyroid (21-4-3) trained with VSS: MSE (red) and classification accuracy (blue) on training set, length of weight vector $\boldsymbol{W}$ (black), absolute value of single weight change $|dw|$ (yellow), MSE decrease due to a given change dE (green). All values are rescaled to fit the plot. (see chapter 3.2.12.5 for the Thyroid dataset description)

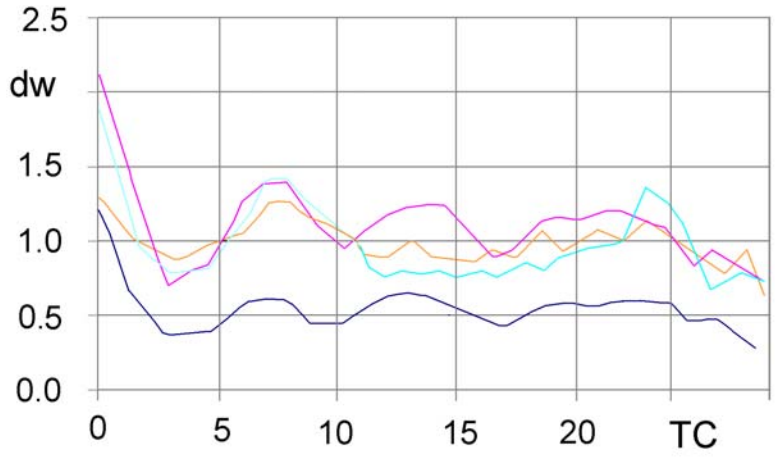

Fig. 2.24. Mean values (M) and standard deviation (S) of hidden (H) and output (O) weight changes during the Thyroid dataset (21-4-3) training with VSS.

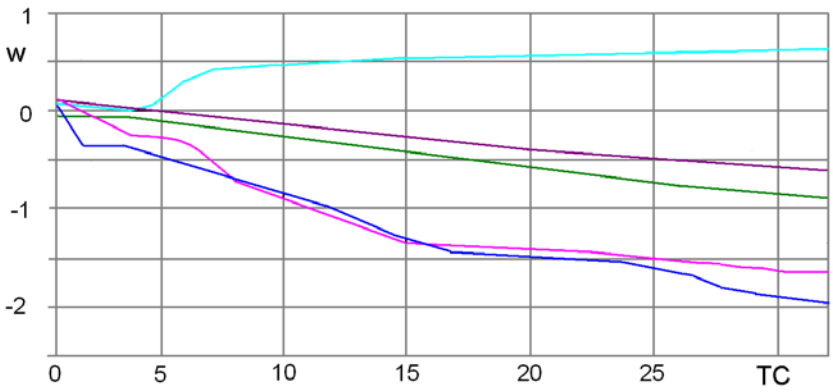

Fig. 2.25. Selected output layer weights. Thyroid (21-4-3), training with VSS.

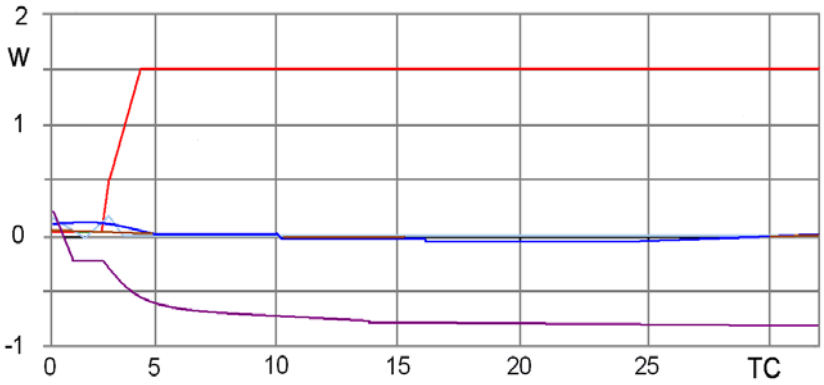

Fig. 2.26. Selected hidden layer weights (among the 8 input features only 2 are meaningful) Thyroid (21-4-3), training with VSS.

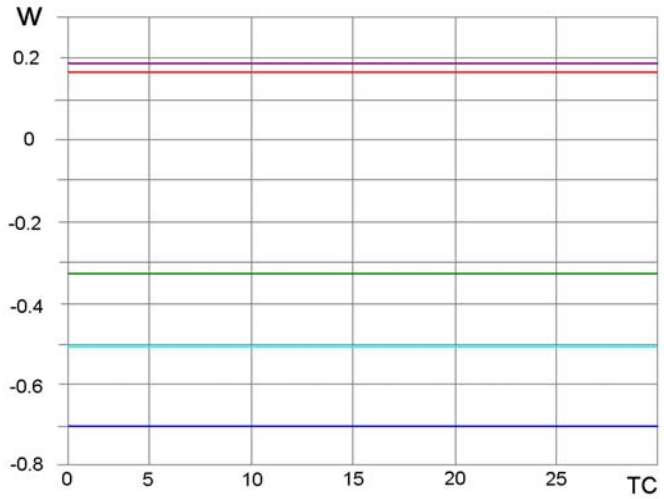

Fig. 2.27. Selected hidden weights of irrelevant features. Mushrooms (125-8-2), training with VSS. (see chapter 3.2.9.2 for the Mushrooms dataset description)

In many datasets, especially in those with large number of features, only some features are useful for classification purposes. The weights connecting hidden layer neurons with the irrelevant features do not change during the training with NG or VSS. Thus, it is very easy to detect the irrelevant features and to remove them from the further training by pruning their weights after the first or second training cycle. Mushrooms and Thyroid are examples of datasets with plenty of irrelevant features (Fig. 2.26, 2.27).

Observation of the weight changes shows that after several training cycles many weights do not change significantly any more and the further training concentrates on adjusting only the values of a few weights (Fig. 2.23). Since some of the weights changed at the beginning of the training, it cannot be assumed that they are irrelevant, but rather that they have already reached their optimal values and these weights can be frozen and not modified any more. A threshold for the minimal weight change must be determined or set a priori. If the weight change in a given training cycle is below the threshold, the weight is frozen for 2^n training cycles, where *n* starts from one and is incremented each time the weight value is determined without being changed. If the change is above the threshold, the weight is normally taken into account in the next training cycle.

These methods of weights pruning and freezing, which can be used as well with VSS as with NG, aim at accelerating the training, however they also improve network generalization by removing the connections that transport only residual noise.

The training algorithms based on analytical gradient frequently underestimate the gradient components in the hidden weight directions (chapter 2.3.3). As a result even the hidden weights of a network trained with LM, which assesses the optimal direction much better than BP, grow much slower than when trained with VSS (Fig. 2.28). Thus VSS reaches the optimal hidden weight values much quicker and after some training cycles no further changes are required (after 4 training cycles in Fig. 2.28-right) That is one of the main reasons why VSS requires fewer training cycles than LM. The output layer weights also grow quicker in VSS trainings, but here the differences between LM and VSS are much smaller. In both algorithms the output layer weights grow faster than hidden layer weights in LM, but slower than the hidden layer weights in VSS.

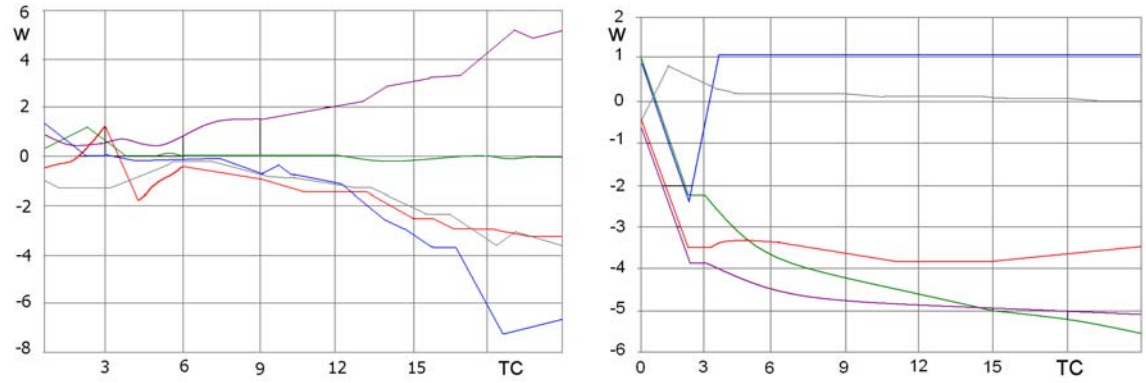

Fig. 2.28. Hidden layer weights for Iris (4-4-3). Left: trained with LM. Right: trained with VSS.

2.4.4. Learning Trajectories

The first and second PCA directions usually capture together about 95-97% of total variance contained in the learning trajectory. Thus, the PCA-based projections of learning trajectories reflect the properties of the original trajectories quite well (Figs. 2.29-2.33).

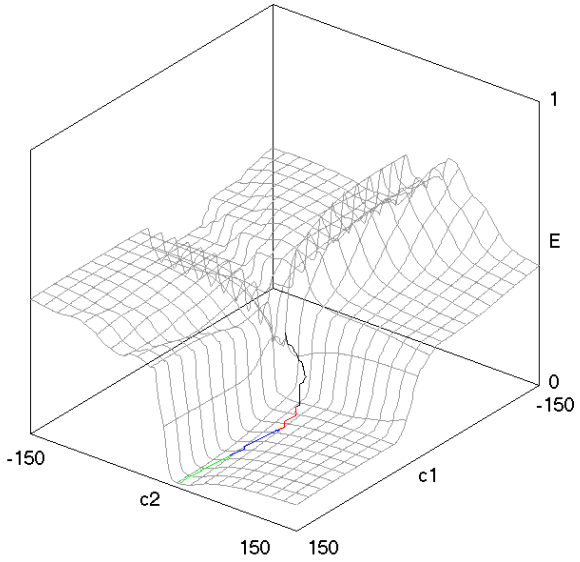

Fig. 2.29. PCA-based projection of Iris (4-4-3) error surface trained with VSS with visible learning trajectory. The trajectory color changes every training cycle.

The trajectories show some regularity for every datasets. Not only *dw* for the same weight in two successive training cycles does not differ much, while *dw* for different weights in the same training cycle may differ ranks of order, but also some trends in weight changes may be observed. All sample plots in this chapter use the same network with 4 inputs, 4 hidden and 3 output neurons trained on the Iris dataset.

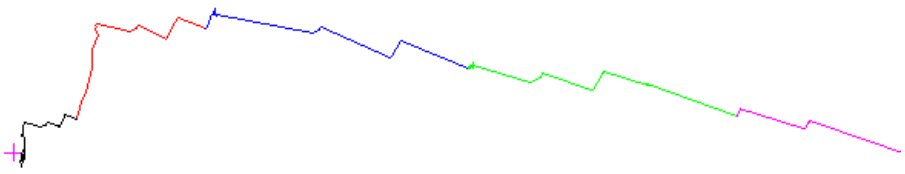

Fig. 2.30. Projection of Iris (4-4-3) learning trajectory trained with VSS in the first and second PCA direction. The cross shows the zero point in the weight space. The trajectory color changes every training cycle.

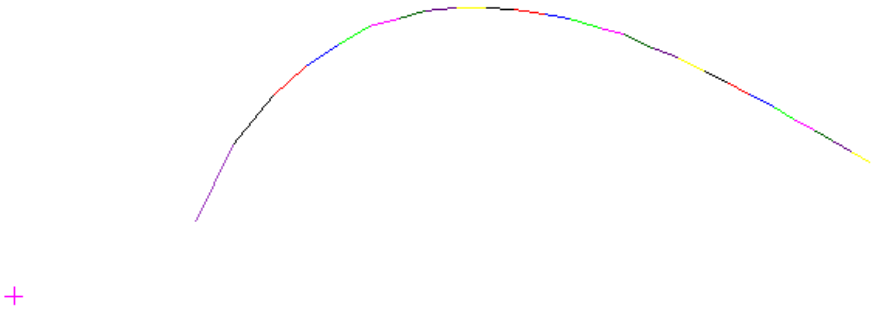

Fig. 2.31. Projection of the Iris (4-4-3) learning trajectory trained with NG without momentum in the first and second PCA direction.

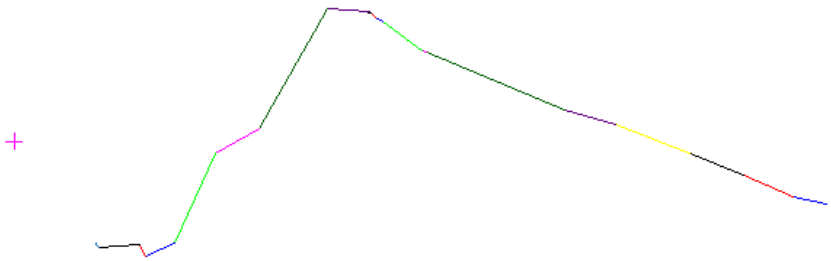

Fig. 2.32. Projection the Iris (4-4-3) learning trajectory trained with LM in the first and second PCA direction.

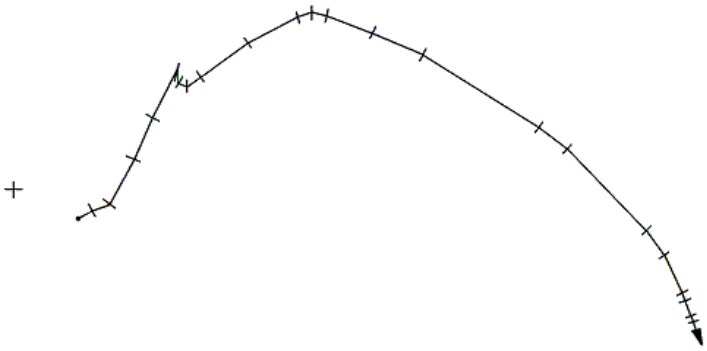

Fig. 2.33. Projection of the Iris (4-4-3) learning trajectory trained with SCG in the first and second PCA direction. The training cycles are divided with short crosswise lines.

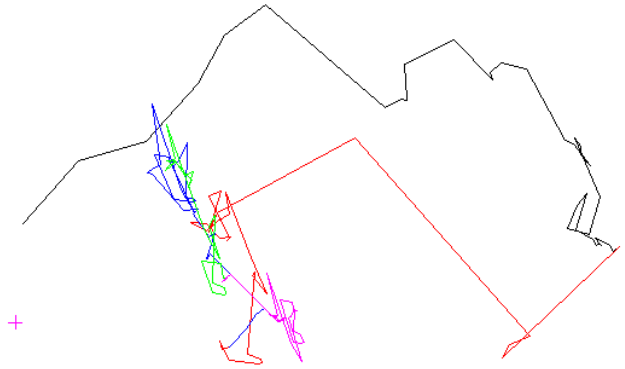

Fig. 2.34. Projection of the Iris (4-4-3) learning trajectory trained with VSS in the third and fourth PCA direction.

Higher PCA components have significant values only at the beginning of the training, what is clear, because at that stage training algorithms chose the proper direction. As the training approaches the final stage, the direction changes are usually slow.

The similarity between all trajectories presented in Figs. 2.30-2.33 is obvious; they create similar arcs following the shape of the Iris error surface ravine. The differences are also clearly visible. Using gradient-based information makes the training dependent on a factor that vanishes as the training progresses, so gradient-based algorithms have a tendency to decrease their learning steps as gradient decreases and thus slowing down the training even more.

VSS does not decrease the step when the gradient decreases, because VSS does not rely on gradient information, but rather on the learning history contained in the trajectory. In general, VSS also sometimes decreases the step, but that is a result of a tighter curvature of the ravine, not of a smaller gradient. VSS stops when the gradient reaches zero values.

2.4.5. Experimental Comparison of VSS, NG, LM and SCG

The numerical experiments were made on some well-known benchmark dataset from the UCI learning repository. The datasets and their detailed description can be found in [Mertz 1998]. Most of the datasets are also described in chapter 3.2.12. For each training algorithm about 20 experiments were made with every dataset. The network was tested on test sets (Thyroid, Shuttle) or in 10-fold crossvalidation (Iris, Wisconsin Breast Cancer, Mushrooms). A vector was considered to be classified correctly if its corresponding output neuron signal was higher then other neuron signals and than 0.5. All training algorithms were run with their default parameters, the same for each dataset. Only sigmoid transfer functions were used, so the additional acceleration of VSS that can be obtained with staircase transfer functions is not revealed here.

Four values determining the algorithm efficiency are considered: the total computational complexity (C_t) required to achieve the desired effect, memory requirements (MB), the quality of the solution the algorithm can find (% accuracy on the test) and the percentage of the algorithm runs that converge to the solution (CR) of this quality.

VSS and NG calculations were done using my own program written in Delphi [Delphi]. Matlab Neural Network Toolbox [NN Toolbox 2004] was used for LM and SCG calculations. For bigger datasets (such as Shuttle) the time of propagating once the training set through the network in NN Toolbox and in my program did not differ more than 5%. For smaller datasets (such as Iris) the times were much shorter in my program. In general, it would not be the best idea to compare the times between Matlab and my program directly. Therefore, the computational complexity of the algorithms was assessed in the following way: first only the datasets were repeatedly propagated through the network with calculating the MSE error N_s times (in the case of Matlab it was done by modifying trainscg.m so that only sim() function was called within the plot). Then the algorithms were run the average number of training cycles require to converge N_t for Mushrooms, Thyroid and Shuttle and the training time T_t was measured. The real training times for Iris and Breast were too short for reliable direct measurement, thus the algorithms were run 1000 training cycles and the measured time T_m was rescaled to T_t: $T_t=T_m(N_t/1000)$. All on-screen display and additional options were switched off in both programs (though for bigger datasets it had negligible influence). A given algorithm computational complexity was calculated for given dataset and network structure per one training cycle as: $C_e=(T_t/N_t)/(S_t/N_s)$.

For VSS C_e was from 18 for the smaller networks (Iris, Breast) to 100 for the bigger networks (Mushrooms). For LM C_e was between 4 and 540 and grew rapidly with network size. For SCG C_e did not depend much on the network size, being between 2 and 8. The number of the training cycles N_t required to converge was always the lowest for VSS and the highest for SCG.

The total computational complexity C_t shown in Table 2.7 reflects the algorithm speed. It expresses the ratio of the total training time to the time of propagating the dataset through the network once. C_t can be obtained by multiplying the per training cycle complexity C_e by the average number of training cycles N_t required to train the network: $C_t=C_eN_t$. It is clear that C_t cannot be calculated very precisely and it will surely vary depending on a given algorithm implementation, nevertheless it provides quite a useful outlook.

In all cases C_t for VSS was lower than that for LM. In most cases, it was also lower than that for SCG, however for larger datasets that are relatively easy to train, such as the Mushrooms dataset, the differences were vanishing.

Only VSS and LM were able to converge to the solutions with the lowest error on the training set (e.g. to classify all training set instances correctly, while the other algorithms made some errors on the training set). However, LM frequently did not converge to the solution and had to be repeated with other starting weights. The CR parameter in Table 2.7 expresses the convergence rate of algorithms, i.e. the percentage of the algorithm runs that converged to the desired solution within 5000 cycles.

For VSS and NG the minimum and maximum number of training cycles in that a given algorithm converges to a given solution differed less than 30% from the mean number N_t given in Table 2.7, while for LM the difference was often over 100%. VSS and NG algorithms had the smallest memory requirements. The performance of NG was poorer than that of VSS. The main difference between the algorithms is that NG uses directly gradient information, while VSS does not.

Additional techniques such as weight freezing, weight pruning, calculating the error not on the entire dataset each training cycle (semi-batch or on-line training) or eliminating vectors that give the least error, lead to much shorter training times with each of the examined algorithms, but since the techniques can be used with all the compared algorithms they are not included here. The methods will be shortly discussed in chapter 2.5.

Table 2.7. Comparison of VSS, NG, LM and SCG algorithms.

dataset	%	VSS				NG				LM				SCG			
network	test	N_t	MB	CR	C_t	N_t	MB	CR	C_t	N_t	MB	CR	C_t	N_t	MB	CR	C_t
Iris 4-4-3	96.0	3.5	-	100	62	11	-	100	175	20	-	80	223	118	-	90	948
Breast 10-4-2	96.0	1.5	-	100	45	4	-	100	112	20	1.5	100	109	147	0.4	60	271
Mushrooms	98.0	1.2	0.4	100	124	21	0.4	100	1070	4	240	90	2180	20	40	100	167
125-4-2	99.6	2.0		100	206	-		0	-	6		90	3260	45		100	377
Thyroid	97.0	6.1	0.2	100	392	40	0.2	40	862	25	30	50	1640	103	1.0	70	581
21-4-3	98.0	10		100	643	-		0	-	35		40	2300	-		0	-
Shuttle	98.0	4.5	1.6	100	423	34	1.6	90	1300	14	1400	60	1430	780	20	50	1480
9-6-7	99.0	6.0		100	564	58		90	1740	19		60	1940	1620		30	3080

N_t - number of training cycles
MB - memory usage in MB for storing network and training parameters, without memory used for the dataset (calculated by subtracting the memory used by the program running the algorithm on a given dataset from the memory used by the program with the given dataset loaded in memory and running the algorithm on the Xor dataset. Memory usage was measured with Task Manager)
CR – convergence rate (percentage of training runs that converged to a given accuracy within 5000 training cycles)
C_t - total computational complexity (ratio of the total training time to the time of propagating the dataset through the network once)

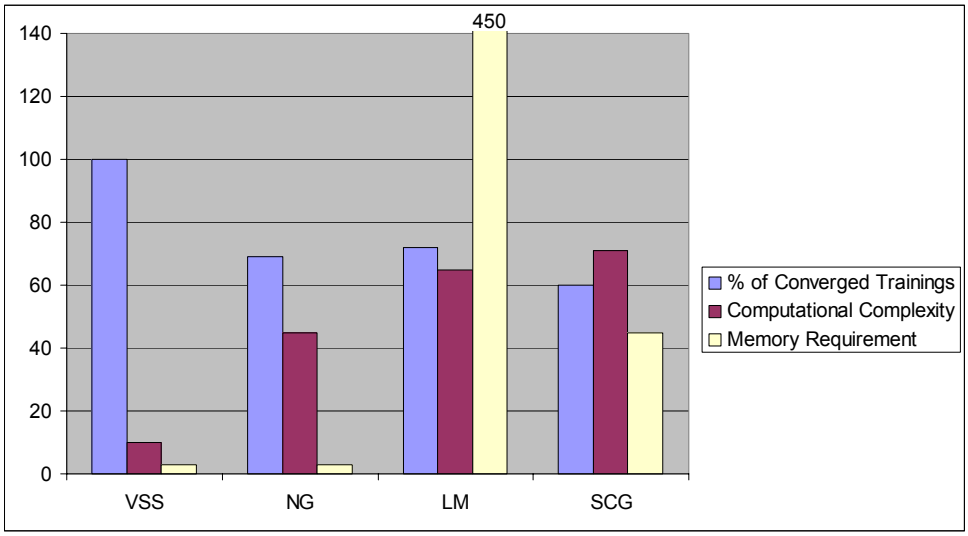

Fig. 2.35. Comparison of VSS, NG, LM and SCG algorithms (mean values from Table 2.7).

VSS does not converge in 100% runs for every dataset (see next chapter). It also did not outperform in every case the other algorithms so much as it could be concluded from the chart above. The chart is made for average values. Thus, the general tendencies shown in the plot below may reflect more faithfully the performance of VSS.

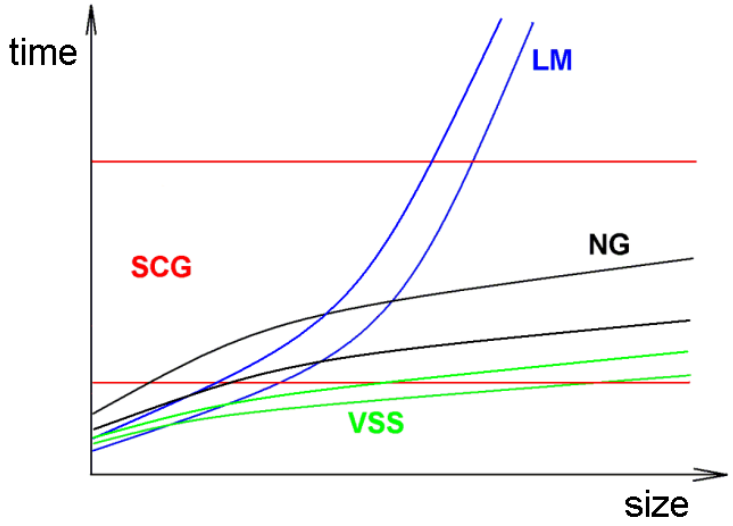

Fig. 2.36. Comparison of VSS, NG, LM and SCG algorithms. General tendencies of relative training times in function of network and dataset size (upper bound for difficult, lower for easy to train datasets).

2.4.6. N-bit Parity Problems

The n-bit parity problems (chapter 1.2.5.2) are very difficult for MLP training algorithms. The following plots of MSE and accuracy on the training set in the function of training cycle show typical VSS performance on n-bit parity problems.

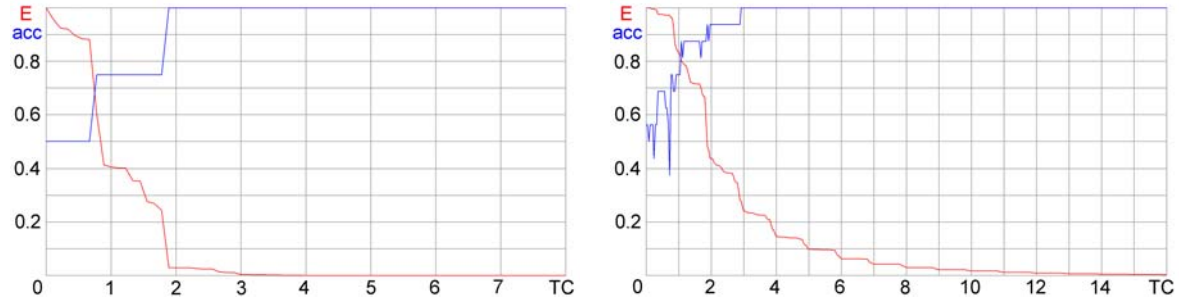

Fig. 2.37. MSE (red) and training accuracy (blue) during the VSS training of: left: Xor (2-2-1), convergence rate ≈ 90%, right: 4-bit parity (4-4-1), convergence rate ≈ 98%.

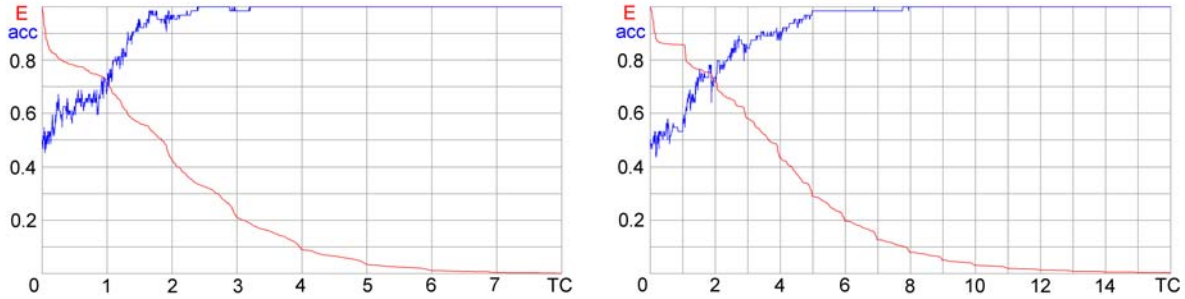

Fig. 2.38. MSE (red) and training accuracy (blue) during the VSS training of: left: 6-bit parity (6-32-1), convergence rate=100%, right: 6-bit parity (6-16-1), convergence rate ≈ 95%.

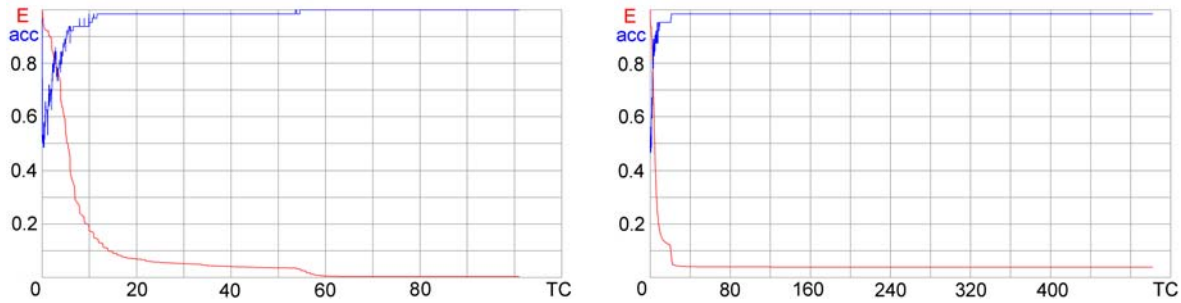

Fig. 2.39. MSE (red) and training accuracy (blue) during the VSS training of 6-bit parity (6-8-1). Left: successful training. Right: two vectors wrongly classified. In this case the convergence rate is about 35% but the accuracy of at least 96.88% (two vectors wrongly classified) is obtained in about 95% of the algorithm runs.

There are $2^{(2-1)}$=4 data clusters per class for the Xor (2-bit parity) problem, $2^{(4-1)}$=8 clusters per class for 4-bit parity and $2^{(6-1)}$=32 clusters per class for 6-bit parity. If there are 32 hidden neurons for the 6-bit parity problem, then the number of hidden neurons equals the number of data clusters per class and the network training is quite easy. Also 32 hidden neurons are required in this case for the SMLP network (chapter 3.2) to describe the 6-bit parity problem with logical rules – each hidden neuron generates one partial rule and the output neuron joins the partial rules with the OR operator. With only 8 hidden neurons the

representation of particular data clusters is distributed among them using complex dependencies, which are difficult to obtain in the network training, since the ravines on the error surface containing the global minima are very narrow (Fig. 1.17-left). In this case the VSS training converges to 100% accuracy in only about one third of the runs, depending on the starting point.

2.4.7. Conclusions

It is clear that search-based techniques, popular in artificial intelligence and completely neglected in neural networks (with an exception of rarely used Alopex algorithm based on simulated annealing), may be the basis for network training algorithms. They may be used for initialization and in combination with traditional gradient-based techniques. However, so far the performance of VSS as a standalone algorithm has been more than satisfactory. It is fast, can find very good solutions and has low memory requirements. Since VSS is very simple to program (does not require calculation of derivatives and matrices), it is quite surprising that in empirical tests it usually outperforms both LM and SCG.

For the error surfaces of real-world datasets local minima in craters are extremely rare. Local search algorithms based on analytical gradient that do not have direct access to the influence of hidden layer weights on the network error cannot precisely determine the gradient direction and fall in spurious local minima. VSS does not fall in spurious minima and seldom requires multistart, only in that case when there is really no downward way from the starting point to one of the global minima.

Although local optimization methods including VSS do not guarantee finding a global minimum for every problem, for the prevailing number of real-world problems they are sufficient and it is rarely required to use global optimization methods, which on the one hand have greater chance to find the solution for complex problems but on the other hand require much higher computational effort [Matthews 2000].

2.5. Decreasing Training Time

The methods of decreasing training time and improving generalization are outlined here because of their importance, though in most cases they can be used with any MLP training algorithms, not only with the search-based ones.

2.5.1. Border Vectors

Neural networks are usually trained on all available data. Support vector machines start from all data but near the end of the training use only a small subset of vectors near the decision borders. The same learning strategy can be used with neural networks, independently of the actual optimization method used. The threshold for acceptance of vectors useful for training is dynamically adjusted during learning to avoid excessive

oscillations in the number of support vectors. Benefits of such an approach include faster training, identification of small number of support vectors near decision borders and may also include higher accuracy of the final solution. Moreover, for strongly imbalanced datasets (with small number of samples in some classes and large in other classes) the solution may be significantly better, automatically focusing on the same number of different classes vectors near the decision borders.

The goal of the Support Vector Neural Training algorithm [Duch 2004b] is to reduce the amount of training data, finding only those training vectors that are really needed to support the training process. Network weights are updated after presentation of the training data, depending on the difference between the target output values and the achieved network outputs. Patterns that are close to the decision borders give significant errors and should be used for further training. If a given pattern contributes to the error less than the threshold, then it is removed from further training.

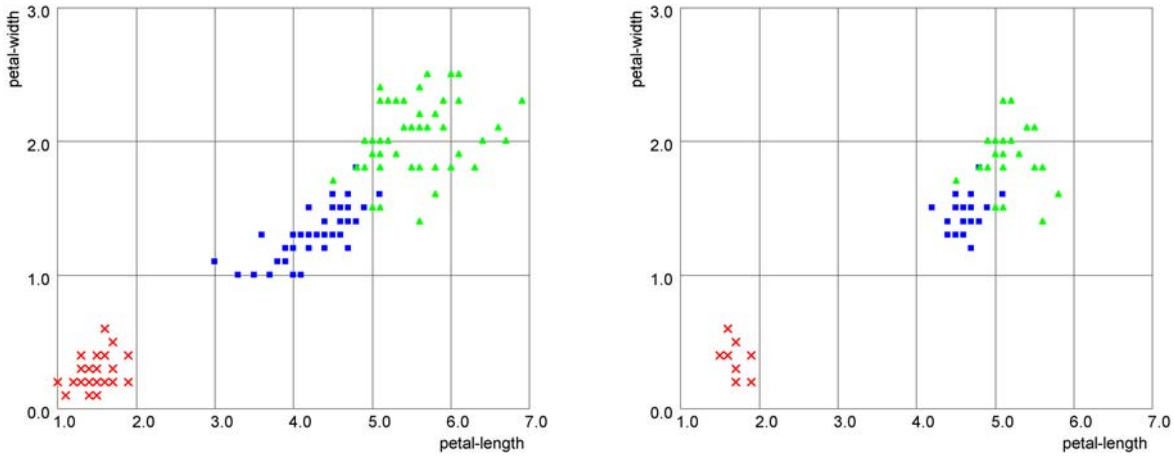

Fig. 2.40. Training vectors of the Iris dataset projected into two most significant input space features. Left: the entire training set. Right: vectors with the greatest error selected for further training.

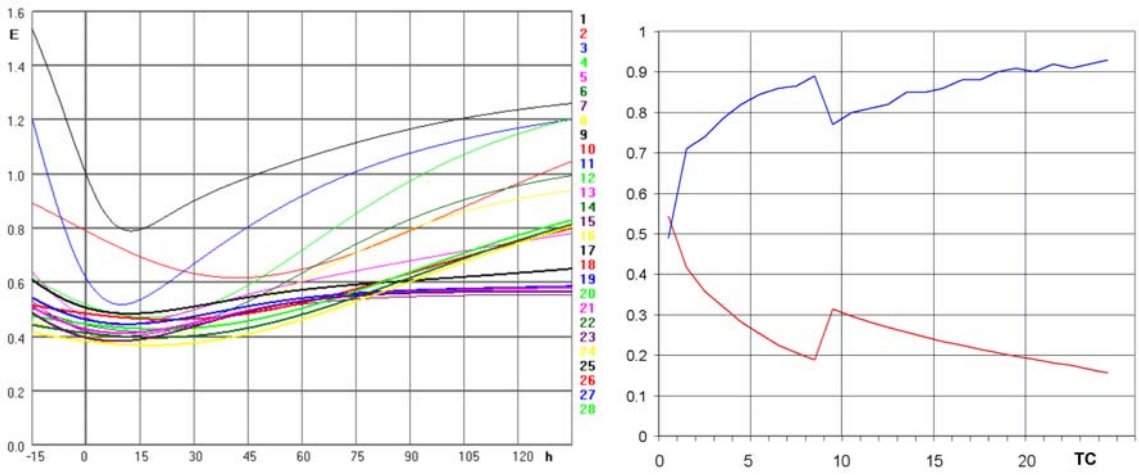

Fig. 2.43. Left: Error surface sections get flatter after the 8th training cycle when border vectors are selected. Right: MSE and classification accuracy on the training set.

There is however one risk of such an approach. If the algorithm is not controlled carefully and the data are noisy, the classification process may invert the decision borders. There are two ways to prevent this: either to use efficient schemes of updating the threshold values [Duch 2004b] or to cluster the vectors with the lowest error instead of rejecting them.

If some vectors are represented by points that lie very close to the proper output space hypercube vertex, they can be clustered and replaced by a single vector. This vector represents the cluster and its error is be multiplied by the number of that cluster instances. That guarantees that the decision borders will not be inverted.

2.5.2. Batch Versus Online Training

Weights can be updated after the entire training set is presented and the error is calculated on the entire set (batch training). In order to decrease training times, only some of the vectors can be propagated through the network and after the partial error is calculated the weights can be updated (semi-batch training). In on-line training, the weights are updated after each single vector is presented. In the examples below, the training set is divided into 10 parts for the Iris and 100 parts for the Mushrooms dataset. Every training vector is randomly assigned to one of the parts at each training cycle.

Table. 2.8. Computational effort reduction of NG training obtained by dividing the training set into parts.

dataset	number of parts	training time reduction (training time with calculating error on the whole data set = 1)
Iris	10	0.30
Wisconsin Breast Cancer	10	0.18
Mushrooms	100	0.047

No modification of the weight update step is required with the number of parts in the training set shown in Table 2.8. However, if we decrease more the number of vectors on which the error is calculated at a time (the batch size), then it is required not to go to the minimum in the gradient direction, but to make a shorter step with NG. Similarly with VSS, when the error is calculated only on a few vectors the weight update can be calculated according to the diagram in Fig. 2.22, but then each weight should be updated about a value proportional to but smaller than the calculated one. If the update values are not smaller than the calculated (as well with NG as with VSS), then the weights will oscillate and the network will be unable to converge.

On-line training decreases the training time about a smaller factor than the number of vectors in the training set. A detailed comparison between efficiencies of batch and on-line training using backpropagation was presented in [Wilson 2003]. Selected results from that work are summarized in table 2.9. Though the training time was different, the average generalization accuracy for on-line and batch training was practically the same. The authors

use about 60% of the original datasets for training and the rest for tests. Since they used a different training algorithm and different network structures, their results for Iris, Breast and Mushrooms differ from mine, but the same trend is visible: stronger acceleration is obtained for bigger datasets. The same authors also compared training times for the Digit Speech Recognition database, using various batch sizes. The results they obtained suggest that decreasing the batch size below a certain number of vectors does not cause further training acceleration.

Table 2.9. Selected experimental results from [Wilson 2003]; training time reduction obtained with on-line BP in comparison to batch BP.

dataset	training set size	training time reduction
Iris	90	1.00
Wisconsin Breast Cancer	410	0.71
Mushrooms	3386	0.011
Shuttle	5552	0.010
Ionosphere	221	0.50
average of 26 datasets	1329	0.05

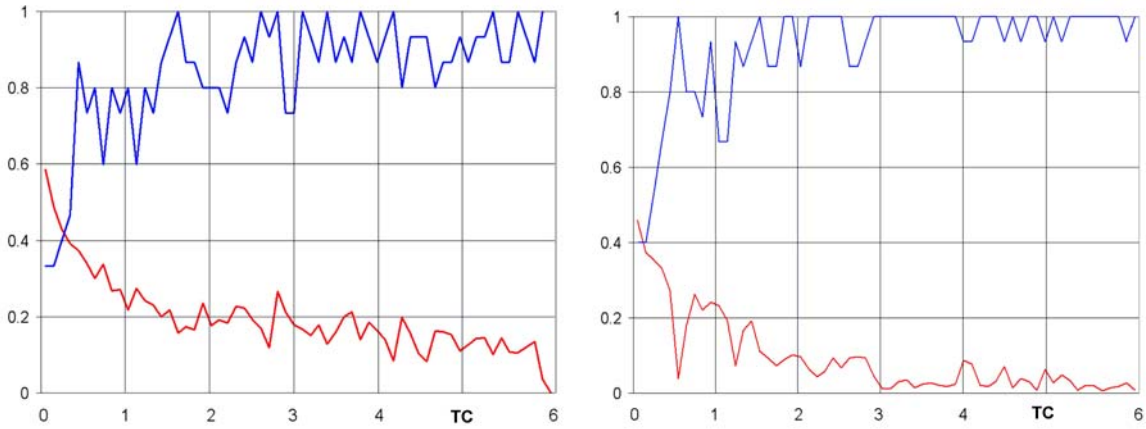

Fig. 2.44. MSE and classification accuracy on (the actual part of) the training set. Training set divided into 10 parts. Iris (4-4-3) trained with: left - standard NG, right - NG with momentum.

Semi-batch training, momentum, border vectors and weight freezing/pruning can be used together in any combination. However, this must be done carefully, since adding each method causes some loss of information. The information cannot be reduced too much, because then the training will not be able to converge. For that reason if some of the methods are combined together, each of them should modify the basic training algorithm less than if used separately (for example the optimal momentum can be 0.4 with batch training and 0.2 with semi-batch training). All the methods work fine with big datasets. If the dataset is small and noisy, efficiency of the methods decreases but for small dataset there is no need for training acceleration.

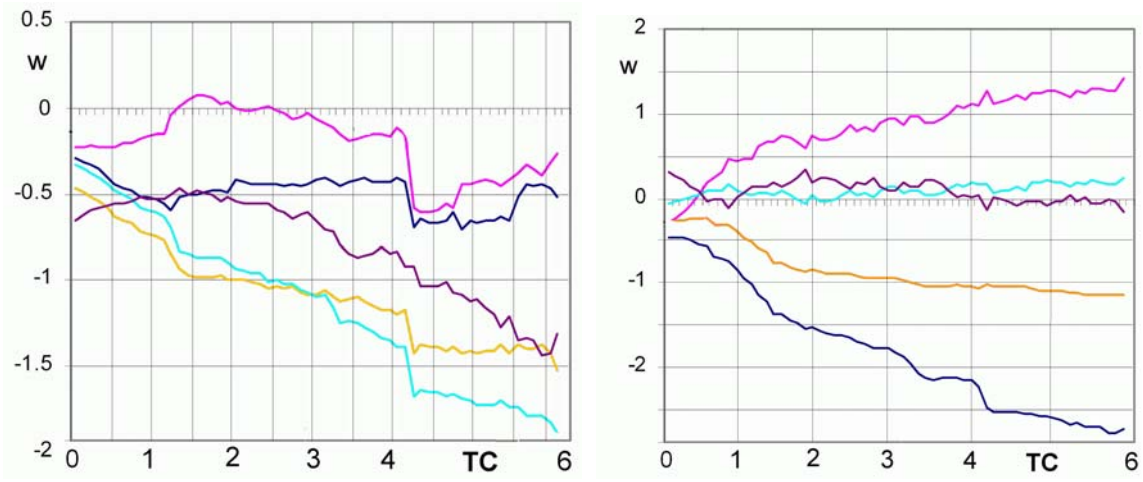

Fig. 2.45. Iris (4-4-3) trained with standard NG. Training set divided into 10 parts. Left: weights of a selected hidden neuron. Right: weights of a selected output neuron.

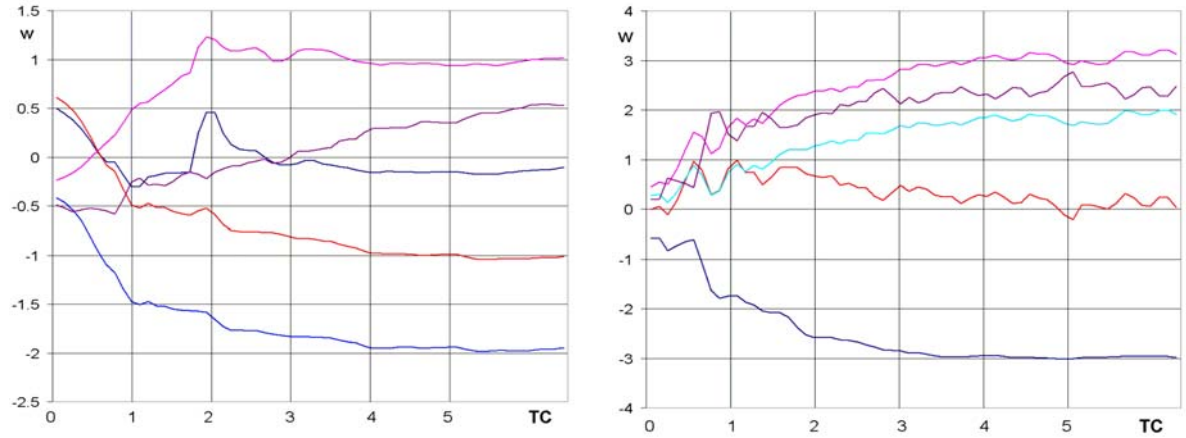

Fig. 2.46. Iris (4-4-3) trained with NG with momentum. Training set divided into 10 parts. Left: weights of a selected hidden neuron. Right: weights of a selected output neuron.

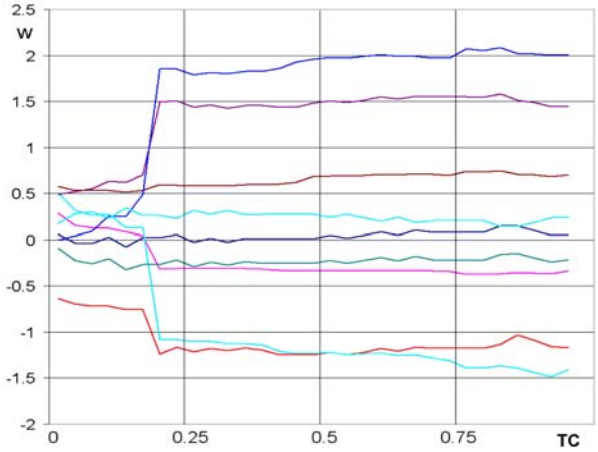

Fig. 2.47. Mushrooms (125-8-2) trained with NG with momentum and 100 parts in the training set. Selected weights of an output neuron.

2.6. Improving generalization

2.6.1. Introduction

Generalization is the neural network ability to learn the data structure and not the single data vectors used for network learning and consequently to make reasonable decisions for data unseen in the learning process. It is known from the approximation theory (Tikhonov regularization) and from the statistical learning theory that too precise learning on a training set leads to overfitting, which results in poor generalization ability [Łęski 2002]. Vapnik-Chervonenkis (VC) theory is a general theory for estimation of dependencies from a finite set of data [Vapnik 1998]. The most important in the VC-theory is the structural risk minimization (SRM) principle. The SRM principle suggests a tradeoff between the quality of the approximation and the complexity of the approximating function. A measure of the approximation function complexity is called VC-dimension (*VCdim*).

VCdim is defined as the number of elements in the greatest set *S*, for which the system can perform all possible 2^n dichotomies of the set (linear divisions of the set into two parts). In the case of a network used for binary classification, *VCdim* equals the maximal number of training vectors that can be correctly reconstructed in all possible configurations. *VCdim* can be assessed as:

$$N_h N \leq VCdim \leq 2N_w(1+\log N_n) \tag{2.42}$$

where N is the dimensionality of input data, N_h is the number of neurons in the hidden layer, N_w is the number of weights in the network and N_n is the total number of neurons. If the sigmoid transfer functions are used, than according to [Hush 1993], *VCdim* can be assessed as:

$$VCdim=2N_w \tag{2.43}$$

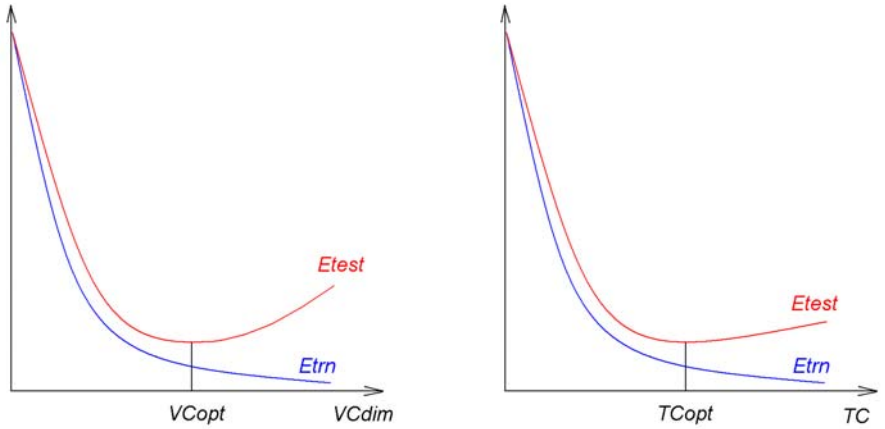

Fig. 2.48. Two factors determining generalization: network complexity corresponding to *VCdim* (left) and number of training cycles (right).

It is usually difficult to design an optimal network structure before the training, especially in situations, where a complicated problem must be solved, and the system must make optimal use of a limited amount of training data. It is known from theory [Denker 1987] and experiments that for a fixed amount of training data, networks with too many weights do not generalize well. On the other hand, networks with too few weights will not have enough power to represent the data accurately (Fig. 2.48-left). The best generalization is obtained by trading off the training error and the network complexity.

The network complexity should correspond to the complexity of the problem the network must solve [Jankowski 1999]. The first choice for the number of hidden neurons may be the geometric mean of the input and output neuron numbers. However, if the data is simple then fewer hidden neurons or no hidden neurons at all will be optimal while for complex data more hidden neurons must be used. Not only the number of neurons should be properly selected but also the fully connected network is not always optimal and some weights can frequently be removed. A simple method of removing irrelevant weights was discussed in chapter 2.4.3. The purpose of that method was rather decreasing training times, although it also leads to improvement in network generalization.

The ideas of some popular methods aiming at improving generalization are presented below. Since the methods can be used with many training algorithms, not only with the search-based ones, they will be only shortly outlined.

2.6.2. Early Stopping

The idea is to use two datasets, one for training and one for validating the generalization performance. Typically, both the training and validation errors will decrease initially but the validation error will start to increase at some point (Fig. 2.48-right). Thus, the training should be stopped when the error on the validation set starts to increase.

This can be explained in two ways. The first explanation (maybe better suited for networks trained for regression problems) is that network learning typically starts from small random weights. This corresponds to simple, essentially linear mappings. As the training proceeds, the weights grow and the network mappings become increasingly nonlinear, i.e. the model complexity grows.

The second explanation is that first all neurons try to solve the task, which mostly reduced the network error, and then the remaining tasks as presented in the figures below. However, from the generalization point of view it is not always desired to solve all the remaining tasks.

The network with 2 inputs (corresponding to X and Y in Fig. 2.49-left), 20 hidden units and 1 output is trained on the dataset shown in Fig. 2.49-left. The aim of the training is to obtain the following network output signal:

0 for the instances shown in red
1 for the instances shown in blue

The three remaining figures show the network output (vertical axis) corresponding the particular points of the input space area (the output signal value 0.5 corresponds to the decision border). After 3 training cycles of VSS the network is in the optimal state, though two training vectors are still misclassified. After 10 training cycles, the accuracy on the training set is 100%, but it is obvious looking at Fig. 2.50-right that such a network has poor generalization abilities.

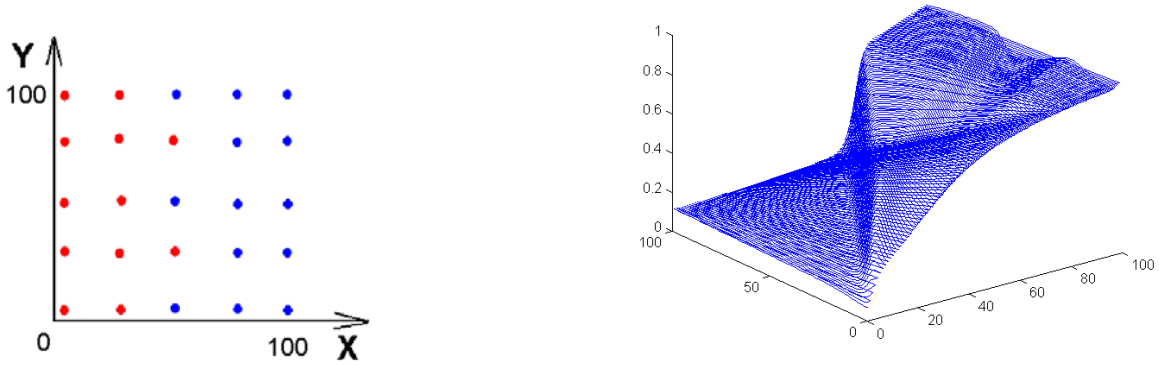

Fig. 2.49. Left: class distribution of the training set. Right: decicion borders after 1 training cycle of VSS (84% accuracy on the training set).

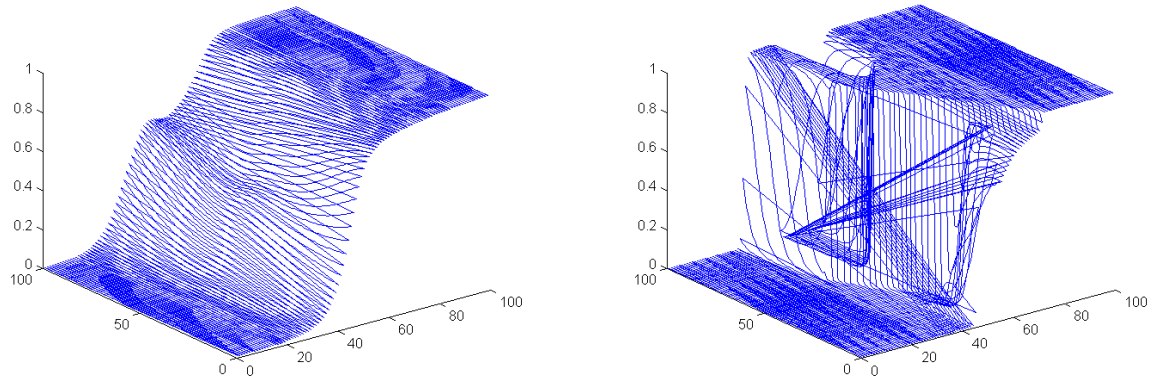

Fig. 2.50. Left: decision borders after 3 training cycles of VSS (92% accuracy on the training set). Right: decision borders after 10 training cycles of VSS (100% accuracy on the training set).

2.6.3. Weight regularization

One technique to reach this tradeoff between the training error and the network complexity is to minimize the cost function composed of two terms: the ordinary training error, plus some measure of the network complexity. The effect of using weight regularization is similar to that of early stopping.

In the simplest weight decay model, the penalty term for big weight values is added to the error function as the sum of all the weight squares. The error function is:

$$E = \sum_{v} \sum_{c} f(d_{v,c} - s_{v,c}) + c \sum_{i} w_i^2 \tag{2.44}$$

As a result, the error surface lifts up (Fig. 1.23), less near the center and more further from the center, thus we can see a superposition of the original ES with the paraboloid caused by the regularization term. It is obvious that the weights will not grow much in this situation.

Nevertheless, this quadratic regularization term has one disadvantage. It influences all weights with the same strength, while frequently the best results can be obtained if some weights are allowed to grow to relatively high values and the others are set to zero [Jankowski 1999].

To solve this problem a weight elimination method was proposed [Weigend 1990, 1991], where the regularization term added to the error function is:

$$c \sum_{i} \frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2} \tag{2.45}$$

in this case, the limit of the regularization term for a single weight is *c* and not infinity, as in the standard weight decay regularization form.

2.6.4. Stretched Sigmoids and Desired Output Signals 0.1 and 0.9

At the final stage of MLP trainings the weights of output layer neurons tend to grow to very high values. This is caused by the sigmoidal transfer function properties (Fig. 1.2-a,b). To obtain zero error, the output neuron signals must be zero or one (-1 and +1 in the case of hyperbolic tangent). This is possible only with infinite weighted sum of the neuron inputs and that forces the infinite growth of weights. To improve network generalization and to prevent the training algorithm from wasting time for the excessive increase of output neuron weights, achieving the training goals must not require infinite weight values. One possibility is to use a stretched sigmoid (Fig. 1.18.b) or other transfer functions that reach the training target value for a finite argument [Duch 1999b]. Another possibility is to set the targets as 0.1 and 0.9 instead of 0 and 1.

2.6.5. ε-insensitive Learning

The ε-insensitive loss function has the following form:

$$E = \max(0, E - \varepsilon) \tag{2.46}$$

Roughly, the idea of this method is that the error must decrease at least by ε to accept the change of parameters leading to the error decrease. The ε-insensitive learning applied to neuro-fuzzy models was considered in [Łęski 2002]. Since neuro-fuzzy models can perform

thinking tolerant to imprecision, but neural network learning methods are zero-tolerant to imprecision, this can remove the inconsistency thus leading to better generalization. The insensitive threshold t will be further used in this thesis to improve classification rules produced by SMLP networks (chapter 3.2).

2.6.6. Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS)

The basic idea of OBD is that it is possible to take a perfectly reasonable network, delete half (or more) of the weights and achieve a network that works just as well, or better [LeCun 1990]. The saliency of a weight is defined as the change of the error function caused by deleting the weight. A simple strategy consists in deleting weights with small saliency. It can be observed that frequently small weights have the least saliency, so a reasonable initial strategy is to train the network and delete small weights. Then the network should be retrained. This procedure can be repeated iteratively.

The main point of OBD is to move beyond the approximation that magnitude equals saliency and propose a saliency measure that uses the second derivative of the error function with respect to the weights. The error function can be approximated by Taylor series:

$$dE = \left(\frac{dE}{dw}\right)^T \cdot dw + \frac{1}{2} dw^T \cdot \frac{d^2 E}{dw^2} \cdot dw + O(\| dw \|^3) \tag{2.47}$$

When the training is finished, it can be assumed, that the network is in the error function minimum and the first term of (2.46) can be ignored. Also the terms higher than the second one can be ignored. Only the second term (Hessian $H = \frac{d^2 E}{dw^2}$) is important. LeCun assumed that only the Hessian diagonal is important, so (2.47) can be written as

$$dE = \frac{1}{2} \sum_i H_{ii} dw_i^2 \tag{2.48}$$

The saliency of each weight is defined as

$$s_i = H_{ii} w_i^2 \tag{2.49}$$

The OBD procedure can be carried out as follows:

1. choose a network architecture
2. train the network until a reasonable solution is obtained
3. compute the second derivatives H_{ii} for each weight
4. compute the saliencies s_i for each weight
5. sort the parameters by saliency and delete some low-saliency parameters
6. go to step 2.

Optimal Brain Surgeon [Hassibi 1993] also uses only the second term in the Taylor series (Hessian). The weight saliency in OBS is:

$$s_i = \frac{w_i^2}{2H_{ii}^{-1}} \tag{2.50}$$

and after the selected weights are pruned all remaining weights are modified about the value dw_i:

$$dw_i = \frac{w_i}{2H_{ii}^{-1}} H^{-1} I_i \tag{2.51}$$

where I_i is a vector consisting of one at the i-th position and zeros elsewhere.

2.6.7. Statistical Weight Analysis

The statistical approach to weight pruning is based on cumulating the differences among different weights in one epoch [Finnhoff 1993][Cottrell 1995]. The weight saliencies are defined as

$$s_i = \frac{| w_i + mean(dw_i^j) |}{std(dw_i^j)} \tag{2.52}$$

where w_i is the weight value before the actual epoch, dw_i^j is the change of the weight w_i as a response to the presentation of the j-th training vector, $mean(dw_i)$ is the mean value and the $std(dw_i)$ is the standard deviation of all the weight changes in the actual epoch. The value s_i is large if the weight is large and its changes are small, otherwise s_i is small and the weight is supposed to be relatively useless.

2.6.8. Growing Networks

Another approach to trading off the training error and the network complexity can be obtained by starting with a very small network and then adding gradually neurons as required. This constructive approach is used by many algorithms [Fahlman 1990][Jankowski 1999, 2003][Adamczak 2001], also by the SMLP network presented in chapter 3.2. If the network without a hidden layer is not sufficient, then the hidden neurons can be added one by one until the results are satisfactory. That can be realized in several ways.

Perhaps the best-known network-growing algorithm is the cascade correlation [Fahlman 1990], which adds the hidden neurons using cascade connection. The network is able to fit perfectly into the training data with limited number of neurons, however the results with crossvalidation or on test sets are not better than for other classification algorithms.

A method used by SMLP networks is described in detail in chapter 3.2. The SMLP network has a separate hidden neurons assigned to particular classes. It starts with one hidden neuron per class and the others are added as needed.

Part 3

Logical Rule Extraction from MLP Networks

3.1. Review of Rule Extraction Algorithms

3.1.1. Decision Trees

3.1.1.1. Introduction

Decision trees are a form of recursive partitioning [Lewis 2000]. Each node can be split into two or more child nodes, in which case the original node is called a parent node. "Recursive" means that the partitioning process can be applied repeatedly. Thus, each parent node can give rise to child nodes and, in turn, the child nodes can split themselves into two further nodes.

The attractiveness of tree-based methods is in large part due to the fact that decision trees represent rules by their nature [Ho Tu Bao 2002]. Therefore, the explanation of any particular classification or prediction is relatively straightforward. Decision-tree building algorithms have the ability to clearly indicate best splits. They put the split that divides into classes the largest number of training records at the root node of the tree. The second strength is that decision trees can deal with continuous and categorical variables. Categorical variables pose problems for some neural networks and statistical techniques. Discretization of continuous features by decision trees is a by-product of applying the splitting criteria in the process of tree building.

There are also many weaknesses of decision tree methods. Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous variable and for time-series data. Some decision-tree algorithms can only deal with binary-valued target classes, others are able to assign records to an arbitrary number of classes, but are error-prone when the number of training examples gets small. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting attribute must be sorted before its best split can be found. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared. Most decision-tree algorithms are univariate, examining only a single feature at a time. This leads to hyperrectangular

decision borders that may not correspond well with the actual distribution of points in the class space.

3.1.1.2. CART

CART (classification and regression tree) [Breiman 1984], is a binary decision tree algorithm, which has exactly two branches at each internal node. The idea of impurity used in CART is formalized in the GINI index for the current node *c*:

$$GINI(c) = 1 - \sum_j p_j^2 \tag{3.1}$$

where p_j is the probability of class *j* in node *c*. For each possible split the impurity of the subgroups is summed and the split with the maximum reduction in impurity is chosen. For ordered and numeric attributes, CART considers all possible splits in the sequence. For *n* values of the attribute, there are *n*-1 splits. For categorical attributes CART examines all possible binary splits. For *n* values of the attribute, there are 2^{n-1}-1 splits. At each node CART searches through the attributes one by one. For each attribute it finds the best split. Then it compares the best single splits and selects the best attribute of the best splits.

CART analysis consists of four basic steps [Lewis 2000]. The first step consists of building a tree using recursive splitting of nodes, during which each node is assigned a predicted class in a way that minimizes the a priori given misclassification costs. The second step consists of stopping the tree building process. At this point a maximal tree has been produced, which probably greatly overfits the information contained within the learning dataset. The third step consists of tree pruning. CART treats pruning as a tradeoff between two issues: getting the right size of a tree and accurate estimate of the true probabilities of misclassification. This process known as minimal cost-complexity pruning results in the creation of a sequence of simpler and simpler trees, through gradually cutting off the increasingly important nodes. The fourth step consists of optimal tree selection, during which the tree that fits the information in the learning dataset, but does not overfit the information, is selected from the sequence of pruned trees.

3.1.1.3. ID3

ID3 algorithm selects which attribute to test at each node in the tree, according to the information gain (entropy). The information gain measures how well a given attribute separates the training examples according to their target classification. ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree [Quinlan 1986][Mulawka 1996].

Entropy that measures homogeneity of examples (characterizes the purity of an arbitrary collection of examples) is used to define information gain precisely. Given a collection *S*, containing positive and negative examples of some target classes, the entropy of *S* relative to the Boolean classification is

$$Entropy = \sum_{i=1}^{nc} (-p_i \log_2 p_i) \tag{3.2}$$

where p_i is the proportion of positive examples in S and nc is the number of classes. In all calculations involving entropy we define 0log0 to be 0. The information gain, *Gain (S, A)* of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{3.3}$$

The central focus of the ID3 algorithm is selecting which attribute to test at each node in the tree, according to the following procedure:

1. See how the attribute distributes the instances.
2. Minimize the average entropy (calculate the average entropy of each test attribute and choose the one with the lowest degree of entropy).

Quinlan [Quinlan 1986] proposed a window-based rule, where only some randomly chosen instances (window) are considered at each iteration step and exception from the generated rules are searched for in the remaining data.

3.1.1.4. C4.5

C4.5 is a an extension of the basic ID3 algorithm designed by Quinlan to address issues not dealt with by ID3 [Hamilton 2002][Quinlan 1986], such as: avoiding overfitting the data (determining how deeply to grow a decision tree), reduced error pruning, rule post-pruning, handling continuous attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handling attributes with different costs and improving computational efficiency.

3.1.1.5. SSV Tree

The SSV (Separability of Split Value) criterion [Grąbczewski 2003] allows to separate objects with different class labels. It can be applied to both continuous and discrete features. The best split value is the one that separates the largest number of pairs of objects from different classes. The *split value* (or *cut-off point*) is defined differently for continuous and discrete features. In the case of continuous features, the *split value* is a real number, in other cases it is a subset of a set of alternative values of the feature. In all cases the left side (*LS*) and right side (*RS*) of a split value s of feature f can be defined for a given dataset D.

$$LS(s, f, D) = \{x \in D : f(x) < s\} \quad \text{if } f \text{ is continuous} \tag{3.4}$$
$$LS(s, f, D) = \{x \in D : f(x) \in s\} \quad \text{otherwise}$$

$$RS(s, f, D) = D - LS(s, f, D) \tag{3.5}$$

$$SSV(s, f, D) = 2 \cdot \sum_{c \in C} |LS(s, f, D_c)| \cdot |RS(s, f, D - D_c)| - \sum_{c \in C} \min(|LS(s, f, D_c)|, |RS(s, f, D_c)|) \tag{3.6}$$

where C is the set of classes and D_c is the set of data vectors from D which belong to class c. According to the SSV criterion the best split value is the one which separates the maximal number of pairs of vectors from different classes and among all split values that satisfy this

condition – the one which separates the smallest number of pairs of vectors belonging to the same class. For every dataset containing vectors, which belong to at least two different classes, for each feature, which has at least two different values, there exists a split value of maximal separability.

The SSV criterion can be used to build decision trees. Since the SSV criterion easily finds the best split points, the generated trees can be small, and can be converted into a small number of crisp logical rules. The classification trees are built by finding the best split of the dataset (which becomes a node of the tree) and splitting the data into two parts for further recursive analysis.

3.1.2. Neural Networks

3.1.2.1. Introduction

Neural network-based rule extraction algorithms fall into two categories: black-box (global) and decompositional (local) methods.

In black-box methods, the analysis of all the network outputs is performed for different inputs, without analyzing the network weights. The network is used to predict the class of the instance but the rules are extracted by some other methods, e.g. by decision trees.

Decompositional methods analyze fragments of the network, usually single nodes to extract rules. Such networks are based either on sigmoidal functions (step function is the logical limit) or on localized functions. Using step functions, the output of each neuron becomes logical (binary), and since the transfer functions are monotonic and their output values are zero and one, it is enough to know the sign of the weight to determine whether its contribution to activation of a given unit is negative or positive. Rules corresponding to the whole network are combined from rules for each network node.

Andrews [Andrews 1995] introduced the following set of criteria for logical rule extraction from data using neural networks:

1. Expressive Power (IF...THEN rules, fuzzy rules, other rules)
2. Translucency (degree in which the rule extraction algorithm looks inside the network)
3. Portability (how well the rule extraction technique covers the set of available network architectures)
4. Quality
   - rule accuracy
   - rule fidelity (how well the rules mimic the NN behavior)
   - rule consistency (the extend to which equivalent rules are extracted from different networks trained on the same task)
   - rule comprehensibility (readability of rules and size of the rule set)
5. Algorithmic complexity.

3.1.2.2. Validity Interval Analysis (VIA)

An example of global methods is Validity Interval Analysis (VIA) proposed by Thrun [Thrun 1995]. VIA is a generic approach to analyzing the input-output behavior of MLP networks. The key idea in VIA is to attach intervals to the activation range of each neuron (or a subset of all neurons), such that the network activation must lie within these intervals, called validity intervals *I*. VIA checks whether there exists a set of network activations inside the validity intervals. It does this by iteratively refining the validity intervals, excluding activations that are probably inconsistent with other intervals. The obtained rules are prepositional if-then rules, where the precondition is given by a set of intervals for the individual input values and the output is a single target category. Rules of this type can be written as:

if (*input contains in the hypercube I*) then *class is C* (or shortly: $I \rightarrow C$)

Two types of approaches can be distinguished: specific-to-general and general-to-specific. In a specific-to-general approach we start with rather specific rules that are easy to verify and gradually generalize those rules by enlarging the corresponding validity intervals. Imagine one has a training instance that, without loss of generality falls into class *C*. The input vector of that instance already forms a (degenerated) set of validity intervals *I*. VIA applied to *I* will confirm the membership in *C* and hence the single point rule $I \rightarrow C$. Starting with *I* a sequence of more general rule preconditions can be obtained by gradually enlarging the precondition of the rule (i.e. the input intervals *I*) and verifying if the new rule is still a member of its class. In a general-to-specific approach we start from rules like "everything is in class *C*" and then new a rule can be generated by splitting the hypercube spanned by the old rule.

3.1.2.3. TREPAN

The TREPAN [Craven 1996a, 1996b] algorithm combines decision trees with neural networks. Decision trees are induced on the training data, plus the new data obtained by perturbing the training data. The additional training data are classified by the neural network. Nodes in the decision tree are split only after a large number of vectors that fall in a given node have been analyzed. Therefore, this method is more robust than direct decision tree approaches, which suffer from a small number of cases in the deeper branches. The algorithm runs as follows:

1. Take a trained network and a set of training data as inputs
2. As output, produce a decision tree
3. Use the network to label the instances
4. Incrementally add nodes to the decision tree

The function to evaluate node N is $f(N) = reach(N) \cdot (1 - fidelity(N))$, where *reach(N)* is the estimated fraction of instances that reach node *N* and *fidelity(N)* is the extend to which the extracted representations accurately model the network for those instances.

3.1.2.4. RULENEG

The RULENEG algorithm [Hayward 1996] is black-box algorithm for binary attributes based on the idea that a conjunctive rule only holds if all antecedents are true. Thus a systematic negation of antecedents in a hypothesized rule can show, which antecedents have to be true to make the rule true. The network is used to test the hypothesized rule.
The algorithm can be described by the following pseudocode [Neumann 1998]:

```
rule system = empty
for every training example E
        find classification P of the network for E
        if E is not classified by a rule in a rule system
                initialize a new rule for P and E
                for every attribute A in E
                        if negation of A leads to classification P (classification does not depend on A)
                                remove A from the rule
                        endif
                endfor
        endif
endfor
```

3.1.2.5. BIO-RE, Partial-RE and Full-RE

BIO-RE [Taha 1996] [Neumann 1998] stands for Binarised Input-Output Rule Extraction. It is a black-box algorithm that extracts binary rules from any neural network. BIO-RE consists of the following steps:

1. Obtain the output of the network for each possible pattern of input attributes
2. Generate a truth table by concatenating each input pattern with its corresponding net output
3. Generate boolean functions from the truth table

Partial-RE is a decompositional algorithm that consists of the following steps:

1. For each hidden and output neuron order incoming connections according to their weights
2. Find individual incoming connections that cause the neuron to fire, if they exist
3. For a connection between neurons i and j, generate rules IF $i \underset{c_j}{\rightarrow} j$ with believe c_j that is equal of the activation value of the neuron j. Mark the connection as being used in the rule

The search for single strong connections continues until the first one not strong enough to activate the neuron by itself is found. If more detailed information is required, the algorithm looks for combinations of two or more unmarked connections that activate the neuron. Finally, rule antecedents representing hidden neurons are replaced by the corresponding set of input attributes.

Full-RE extracts all possible rules and corresponding certainty factors. For each neuron the following rule is generated: $[w_{1j}x_1 + w_{2j}x_2 + ... + w_{nj}x_n] > \alpha_j \underset{c_j}{\rightarrow} j$, where w denotes the weight and x the input.

3.1.2.6. RX

The RX algorithm [Setiono 1995] [Neumann 1998] runs as follows:

1. Train and prune the NN
2. Discretize the activation values of the hidden units by clustering
3. Generate rules that describe the network outputs using the discretized activation values
4. For each hidden unit:
    a. if the number of input connections is less than an upper bound, then extract rules to describe the activation values in terms of the inputs
    b. else form a subnetwork
        i. set the number of output units equal to the number of discrete activation values. Treat each discrete activation values as a target output
        ii. set the number of input units equal to the number of inputs connected to the hidden units
        iii. introduce a new hidden layer
        iv. apply RX to this subnetwork
5. Generate rules that relate the inputs and the outputs by merging rules generated in step 3 and 4.

3.1.2.7. Subset Algorithms

A number of decompositional approaches such as SUBSET [Towell 1991], KT [Fu 1994], RULE-OUT [Decloedt 1996] and Destructive Learning [Yoon 1994] differ only in some details but share the same technique for the rule extraction process:

For each hidden and output neuron C:

1. Find all combinations p with positive weights to C whose sum exceeds the threshold of C
2. For each $p = \{p_1,...,p_i\}$
    a. find the set S_n of all combinations of negative weights to C, such that the sum of the weights of p and the weights of N-n exceeds the threshold of C, where N is the set of all negative weights for C and n is an element of S_n
    b. for each element $n = \{n_1,...,n_j\}$ create the rule:
        if $p_1,...,p_i$, not $n_1,...,$ not n_j then C

3.1.2.8. M-of-N

To overcome the high complexity of SUBSET and to further increase the comprehensibility of a rule system, Towell [Towell 1991] developed the following M-of-N algorithm:

1. For each neuron, cluster the incoming connections into groups with similar weights
2. Average the weights within each cluster
3. Eliminate the clusters without significant effect on the output of the neuron
4. re-train the network with frozen weights to optimize biases
5. form a single rule for each neuron
6. simplify rules to M-of-N form

3.1.2.9. RULEX

The RULEX [Andrews 1994] algorithm is based on constrained MLP networks with pairs of sigmoidal functions combined to form ridges or local bumps. Rules in this case are extracted directly from an analysis of the weights and thresholds. Disjoined regions of the data are covered by different hidden units. In effect, this method is similar to a localized network with step activation functions. The method works with continuous as well as with discrete inputs

3.1.2.10. NeuroRule and M-of-N3

Neurorule and M-of-N3 are two similar decompositional algorithms developed by Setiono [Setiono 2000a]. They share the common network training and rule extraction technique:

1. Select and train the network to meet the prespecified accuracy requirement
2. Remove the redundant connections in the network by pruning while maintaining its accuracy. Steps 1 and 2 can be repeated several times if required.
3. Discretize the hidden unit activation values of the pruned network by agglomerative clustering (the neighboring activation values of different input patterns are joined together as long as this does not change the network classification)
4. Extract rules that describe the network outputs in terms of the discretized hidden unit activation values (find any combination of hidden neuron signals that causes the output neuron to fire, i.e. to produce the positive output signal)
5. Generate rules that describe the discretized hidden unit activation values in terms of network inputs (find any combination of inputs that makes the hidden neuron activation within particular discretization interval)
6. Merge the two sets of rules to obtain a set of rules that relates the inputs and outputs of the network

Both the hidden and output neuron use hyperbolic tangent transfer functions. The algorithms require discrete input data. The present value of a given feature is coded as +1 and the absent values as –1. The training process starts with an oversized network that is successively pruned. In the case of M-of-N3, after the small weights are removed, the remaining positive weights are set to +1 and the negative ones to –1. Since the network training starts with random weights, different rule sets can be extracted from the same dataset, depending on the initial weights distribution. In the discussion with me, Setiono admitted that in general he considers Neurorule the best of his rule extraction algorithms.

3.1.2.11. FERNN

Since the repetitive network training and pruning is a time consuming process, Setiono proposed an algorithm for ”Fast Extraction of Rules from Neural Networks” (FERNN) [Setiono 2000b], which extracts the rules without weight pruning in the following way:

1. Identification of useful hidden units based on the information contained in these units. For this purpose C4.5 is employed.
2. Identification of relevant connections from the input units to the useful hidden units based on magnitudes of their weights.

Thus it can be said that FERNN is a mixed algorithm: it performs the analysis of the input-to-hidden weights but uses the black-box approach (by employing C4.5) to hidden-to-output weights.

3.1.2.12. FSM

FSM (Feature Space Mapping) [Adamczak 2001] is a constructive neural network that estimates probability density of input-output pairs in each class. The architecture of the FSM network, which is based on the RBF network architecture, consists of three layers (input, hidden and output). The number of nodes in the hidden layer depends on the problem and is found automatically during the training phase. There can be only one node in the output layer, which estimates the confidence of the classification or there can be one output node per class.

Generally, there is no restriction upon the type of transfer functions in the FSM model, however so far only localized functions G, such as gaussian, bicentral, triangular and rectangular, were used:

$$G(x;D,\sigma)=\prod_i G_i(x_i;D_i,\sigma_i) \tag{3.7}$$

Rectangular functions are especially useful for crisp logical rule extraction, other functions lead to fuzzy rules. The FSM network realizes the following function:

$$FSM(x) = class\left(\max_i (G_i(x;D,\sigma))\right) \tag{3.8}$$

The initial structure of the network includes input and output units and a single layer of hidden units with parameters determined by a clustering algorithm [Duch 1997]. For on-line learning, the initialization of additional hidden nodes is performed after a fixed number of incoming training vectors. One of the problems with RBF networks is their inability to select relevant input features. In FSM feature selection is performed by adding a penalty term for small dispersions to the error function.

3.1.2.13. MLP2LN

The MLP2LN network uses the same structure as the SMLP network (Fig. 3.1), however the two networks use quite different training algorithms.

To facilitate extraction of logical rules from an MLP network, one can transform it smoothly into a network performing logical operations – a logical network (LN). This transformation is the basis of the MLP2LN algorithm [Adamczak 2001]. One can try to extract logical rules from an already trained network. However, starting from a single neuron or constructing the LN using training data directly (constructive, or C-MLP2LN algorithm) is

faster and usual more accurate. Since the interpretation of MLP network activation is not easy, a smooth transition from MLP to a logical type of network performing similar functions is advocated. This transition is achieved during network training by the following:

1. Increasing gradually the slope of sigmoidal functions to obtain crisp decision regions.
2. Simplifying the network structure by inducing the weight decay through a penalty term.
3. Enforcing integer weight values –1, 0, 1 interpreted as: 0 = irrelevant input, -1 = negative evidence, +1 = positive evidence. These objectives are achieved by adding two additional terms to the error function:

$$E = \frac{1}{2}\sum_{v}\sum_{c}(y_{c,v} - d_{c,v})^2 + \frac{\lambda_1}{2}\sum_{i} w_i^2 + \frac{\lambda_2}{2}\sum_{i} w_i^2 (w_i^2 + 1)^2 (w_i^2 - 1)^2 \tag{3.9}$$

The first part is the standard MSE measure of matching the network output y with the desired output d for all output neurons c corresponding to particular classes and all training data samples v. The first term scaled by λ_1 is used frequently in the weight pruning or regularization methods to improve generalization of MLP networks. A naive interpretation of why such regularization works is based on the observation that small weights mean that only the linear part of the sigmoid is used. Therefore, the decision borders are rather smooth. On the other hand, for logical rules, sharp decision borders are needed. To achieve these objectives, the first regularization term is used at the beginning of the training to force some weights to become sufficiently small to remove them.

The second regularization term, scaled by λ_2 has a minimum (zero) for weights approaching –1, 0 and +1. The first term is switched off and the second increased in the second stage of the training. This allows the network to increase the remaining weights and, together with increasing slopes of the sigmoidal functions, to provide sharp, hyperrectangular decision borders. Thus, the network is transformed into a logical network by increasing the slope of sigmoidal functions to infinity, changing them into the step functions. Such a process is difficult, since a very steep sigmoidal functions leads to the noncontinuous gradients.

The training can process separately for each output class. A single hidden neuron per class is created and trained using a backpropagation procedure with regularization. λ_1 and the slopes of sigmoidal functions are increased gradually and weights with a magnitude smaller than 0.1 are removed. λ_2 is then increased until the remaining weights reach –1, 0, 1 +/- 0.05. Finally very large slopes (about 1000) and integer weights –1, 0, 1 are set, effectively converting neurons into threshold logic functions. The weights of existing neurons are frozen and new neurons (one per class) are added and trained in the same way as the first ones. This procedure is repeated until all data samples are classified correctly, or until the number of obtained rules grows sharply, indicating overfitting (for example one or more rules per one new vector classified correctly are obtained).

The C-MLP2LN network expands after a neuron is added and then shrinks after connections with small weights are removed. A set of rules is found for each class separately. The output neuron for a given class is connected to the hidden neurons created for that class. In some cases, only one hidden neuron can be sufficient to learn all instances, becoming an output neuron rather than a hidden neuron. Output neurons perform summation of the incoming signals and have either positive weight +1 (adding more rules) or negative weight –

1. The last case corresponds to those rules that cancel some of the errors created by the rules found previously that were too general. They may be regarded as exceptions to the rules.

The network requires discrete inputs. If the data is continuous it must be discretized before giving it to the network inputs. Domain knowledge that can help to solve the problem can be inserted directly into the network structure, defining initial conditions, which could be modified further in view of the incoming data. Since the final network structure becomes quite simple, inserting partially correct rules to be refined by the learning process is quite straightforward.

3.1.3. Fuzzy and Neuro-Fuzzy Systems

The fuzzy modeling is based on the premise that human thinking is tolerant to imprecision, and the real world is too complicated to be described precisely [Łęski 2002]. A neuro-fuzzy system is a fuzzy system trained with some algorithm derived from the neural network domain. The integration of neural networks and fuzzy systems aims at the generation of a more robust, efficient and easily interpretable system where the advantages of both models are kept and their possible disadvantages are removed.

A neuro-fuzzy interference system (NFIS) performs multi-input-single output fuzzy mapping $\boldsymbol{X} \rightarrow \boldsymbol{Y}$, where $\boldsymbol{X} \in \boldsymbol{R}^n$ and $\boldsymbol{Y} \in \boldsymbol{R}$. The main blocks of the NFIS are: fuzzifier, rule base, inference and defuzzifier. The fuzzifier performs a mapping from the observed crisp input space $\boldsymbol{X} \in \boldsymbol{R}^n$ to the fuzzy sets. The fuzzy rule base consists of a collection of N fuzzy if-then rules, aggregated by disjunction or conjunction. The fuzzy inference determines a mapping from the fuzzy sets in the input space $\boldsymbol{X}$ to the fuzzy sets in the output space $\boldsymbol{Y}$. Each of N rules determines a fuzzy set B. The defuzzifier performs a mapping from a fuzzy set B to a crisp point y in $\boldsymbol{Y} \in \boldsymbol{R}$. The training algorithm is based on backpropagation.

3.1.3.1. FLEXNFIS

There are two approaches to NFIS designing: the Mamdani method, where conjunction is used for inference and disjunction to aggregate individual rules and the second "logical-type" method, where fuzzy implications are applied to inference and conjunction to aggregation. Frequently Mamdani-type systems are more suitable to approximation problems, whereas logical-type systems may be preferred for classification problems.

The FLEXNFIS model [Rutkowski 2003] can learn not only the parameters of the membership functions but also the type of systems (Mamdani or logical). Consequently, the structure of the system is determined in the learning process. Several types of FLEXNFIS systems can exist. For example, the AND-type FLEXNFIS is characterized by the simultaneous appearance of Mamdani-type and logical-rule systems, while the OR-type FLEXNFIS depending on a certain parameter exhibits "more Mamdani" or "more logical" behavior.

3.1.3.2. NEFCLASS

The NEFCLASS model [Nauck 1999][Hoffmann 2002] is based on a three-layer fuzzy perceptron network. It uses fuzzy sets as weights between the input and the hidden layer and 0/1 weights between the hidden and the output layer. Input neurons correspond to the features, hidden neurons represent the fuzzy rules and output neurons represent different classes. A fuzzy if-then rule is generated by a hidden neuron by assembling all its connection weights to input layer in the antecedent part and by setting the conclusion part equal to the class of the output neuron to which the hidden neuron is connected.

Prior to the learning process, each feature is equipped with a number of fuzzy sets. The sets are associated with linguistic terms, which in turn form the universe of input to hidden layer connection weights and thus make up the granularity of description for each feature in the antecedent part of the rule. The fuzzy sets can be shifted or their core or support can be expanded or contracted in the learning process, but their connections with the linguistic terms remain fixed.

The rule induction algorithm consists of three parts: 1 – creation of an initial set of rules, 2 – selection of the best rules according to some criterion, 3 – the fine tuning of the fuzzy sets that model the linguistic terms. The third step, called fuzzy backpropagation, uses a fuzzy heuristic variant of the gradient descent method.

3.1.3.3. FuNN

The FuNN model [Kasabov 1996, 1999, 2003] is based on a five-layer feedforward neural network. The first layer of neurons receives the input information. The second layer calculates the fuzzy membership degrees to which the input values belong to the predefined fuzzy membership functions. The third layer of neurons represents associations between the input and the output variables, fuzzy rules. The fourth layer calculates the degrees to which output membership functions are matched by the input data. The fifth layer calculates the exact values for the output variables defuzzification. The number of neurons and connections can be dynamically changed by the training algorithm.

The membership functions used to represent fuzzy values are triangular with the centers of triangles being attached as weights to the corresponding connections. The membership functions can be modified through learning, changing the centers and the widths of the triangles. Several training algorithms, such as backpropagation or genetic algorithms have been developed for FuNN, as well as several rule extraction algorithms.

3.1.3.4. Four-layer Neuro-fuzzy Systems

An interesting four-layer neuro-fuzzy scheme for designing a rule-based classifier along with feature selection was proposed in [Chakraborty 2004], however the authors did not name their solution. The network is trained with backpropagation in three phases. In the first phase, the network learns the important features and the classification rules. In the subsequent phases, the network is pruned to an “optimal” architecture that represents an “optimal” set of rules. The pruned network is further tuned to improve performance.

The first layer consists of input nodes, the second one performs fuzzification and feature analysis, the third one contains antecedent nodes (each node in this layer represents the *if* part of a rule) and the forth one contain the output nodes that represent the *then* part of the rules.

3.1.4. Hybrid Systems

Two hybrid systems in which the neural network is use for the improvement of the input data quality and another system is used for rule extraction are presented here. Neural networks proved to be capable of providing solutions for some cases of medical diagnosis of higher quality than decision trees and some other systems. However, despite their success they had problems with being widely accepted by the medical community due to the lack of transparency in the methods they use to reach the diagnosis. A critical factor in medical diagnosis is the necessity to be able to explain how a diagnosis was reached. Therefore, another system was used to obtain transparent explanations of the decisions. The SMLP network presented in chapter 3.2 aims at joining these two abilities (high accuracy and clear explanation).

3.1.4.1. GEX and GenPar

Methods called GEX and Genoa were proposed in [Markowska 2002] and [Markowska 2004]. An MLP network is used to predict the class of a given data instance. Then the genetic algorithm-based rule extraction module generates rule not for the original class of the actual instance but for the class predicted by the neural network. The advantage of that approach is that the neural network clears the data from noise, thus the rules can be more accurate and comprehensive. At the beginning the chromosome is decoded to a rule set. Afterwards the training patterns are applied to the rule set and the neural network. Each individual is evaluated on the base of accuracy (number of misclassified examples) and comprehensibility (number of rules and premises). Then the algorithm searches for the best individuals and calculates the global adaptation value for each of them. In the last step individuals are drawn to the reproduction and finally by applying genetic operators the new population is produced. This method follows the key idea of the TREPAN algorithm, using instead of decision trees logical rules optimized with genetic algorithms.

3.1.4.2. C4.5 Rule-PANE Algorithm

C4.5 Rule-PANE [Pennigton 2003] is a rule-based machine learning technique that employs a neural network as a pre-process in the organization of a rule set. This technique is believed to provide the strong generalization capability inherent in neural networks, along with the obvious comprehensibility of a rule set. The training dataset is used to generate a neural network ensemble using bagging or boosting. Thus, each network in the ensemble is trained on a slightly different dataset. (Boosting instead of just drawing a succession of independent samples from the original dataset as in the bagging approach, maintains a weight for each instance. As the boosting process progresses, higher weights are given to the data instances that have not been successfully classified by previous networks generated for the ensemble. The higher the weight, the more influence the data item has on the learning process

of the current neural network. In AdaBoost.M1 [Freund 1997] the network error given by the instance is multiplied by its weights. The other alternative is to increase the number of that instance samples in the original dataset proportionally to the sample weight. Still another technique is multiple boosting [Zhengz 1998] that uses ensemble of ensembles to obtain results more accurate than through bagging and more stable than through boosting.) The data items are then passed through the neural networks one last time and the new dataset is created using the classes assigned to the instances by the network ensemble. An additional dataset is then created by randomly generating further data items. The union of the two datasets is used as the training data for C4.5 Rule.

3.1.5. Other Algorithms Used in Comparison of Experimental Results

Besides some of the algorithms presented above the following classification and/or rule extraction algorithms are used in comparisons of the experimental results in chapter 3.2.12:

LVQ (Learning Vectors Quantizers) [Kohonen 1990] is a supervised classification system based on the nearest-neighbor rule, in which a dramatic gain in computational speed can be obtained by reducing the number of vectors that represent each class by clustering. A set of reference vectors, also called codebook vectors, is obtained through an iterative process according to the competitive learning rule (only the closest vector, called winning, moves toward the presented data at each iteration).

SOM (Self-Organizing Map) [Kohonen 1984][Naud 2001] is a particular type of neural networks that combines multivariate data visualization and clustering capabilities. The output layer of neurons is a two-dimensional array (map) that is directly used for data visualization. The learning process is unsupervised and self-organized. It is similar to the LVQ algorithm, but while in LVQ each unit is updated independently, in SOM the winning unit interacts with its neighbor, which are also moved toward the data, the more the closer they are to the winning unit.

AQ15 [Michalski 1995] is a rule induction system that searches for the sets of rules in the discrete feature space. It generates rules describing a single class, using training vectors from that class as positive and others as negative examples. For multi-class problems, it is enough to repeat the algorithm for each class.

CN2 [Clark 1989] is a rule induction system, that modifies the basic AQ algorithm in such a way that it is able to deal with noise and other complications in the data. CN2 does not automatically remove from its consideration a candidate that includes some negative examples. Rather it retains a set of complexes in its search that cover large number of examples of a given class and few of other classes. At each step it either adds a new conjunctive term or removes a disjunctive one. Having found a good complex, CN2 removes those examples it covers from the training set and adds the rule “if <complex> then predict <class>” to the end of the rule list. The process terminates for each given class when no more acceptable complexes can be found.

ITRULE [Goodman 1989] is a rule induction system, which uses a maximum entropy estimator to rank the hypotheses during decision rule construction. It produces rules in the form: "if <all conditions> then <class> with probability 1".

LERS (LERning from exampleS) [Grzymała-Busse 1999] is based on rough sets and uses discrete features. LERS searches for a minimal length description for each class represented in the training set. It generates two sets of rules: certain and possible, respectively for the lower and upper approximation of the set.

1R [Holte 1993] is a decision tree based on single attribute. It allows for discovering simple correlations between features and classes, however in tests it is usually not so accurate as more complex classification algorithms.

AC^2 [Statlog 1994] is not a single algorithm, but rather an expert system, which allows the user to build graphically a decision tree by placing a considerable emphasis on the dialog between the system and the user.

Bayes Tree [Buntine 1993] is a Bayesian approach to decision trees. It is based on a full Bayesian approach: as such it requires the specification of prior class probabilities (usually based on empirical class proportions), and a probability model for the decision tree.

CAL5 [Müller 1997] is a decision tree especially designed for continuous and ordered discrete attributes, though an added sub-algorithm is able to handle unordered discrete attributes as well. CAL5 separates the examples from *n* dimensions into areas represented by subsets of samples, where the class exists with a probability greatest than a given decision threshold. Similar to other decision tree methods, only class areas bounded by hyperplanes parallel to the axes of the feature space are possible.

CASTLE (CAsual STructures from inductive LEarning) [Acid 1991] is a program that implements casual (Bayesian) networks.

DIPOL92 [Statlog 1994] is a learning algorithm, which constructs an optimized piecewise linear classifier by a two-step procedure. In the first step the positions of the discriminating hyperplanes are determined by pairwise linear regression. Then to optimize these positions in relations to misclassified patterns an error criterion function is minimized by a gradient descent procedure for each hyperplane separately.

FACT (Fast Algorithm for Classification Trees) [Loh 1988] uses statistics based on some assumptions about the probability distribution. It divides continuous features and discrete features are converted to continuous ones with special methods.

QUEST (Quick, Unbiased, Efficient, Statistical Tree) [Loh 1997] is the more complex version of the FACT algorithm.

FDA (Fisher's Discriminant Analysis) [Fisher 1936] uses hyperplanes in n-dimensional feature space to separate the known classes as well as possible by optimizing a quadratic cost function. Vectors are classified according to the side of the hyperplane they fall on.

LDA (Linear Discriminant Analysis) [Schalkoff 1992] searches for a hypersurface that separates vectors that belong to two different classes and keeps the maximal distance from both classes.

LogDA (Logistic Linear Discriminant Analysis) [Statlog 1994] operates by choosing a hyperplane to separate two classes as well as possible, where the criterion for a good separation is maximization of the conditional likehood. However, in practice, there is often very little difference between LDA and logDA, and the linear discriminants provide good starting point for the logistic ones that computationally are much more expensive.

NewID is a decision tree algorithm similar to C4.5. It performs probabilistic classification, but unlike C4.5 NewID does not perform windowing [Statlog 1994].

OC1 (Oblique Classifier) [Murthy 1997] searches for decision trees using hill climbing and uses a combination of heuristic and non-deterministic methods to find the linear combinations of features in the tree nodes.

PVM (Predictive Value Maximization) [Weiss 1990] performs a full search in the solution space. It is very efficient for small datasets, however for large datasets it may run into combinatorial explosion problems.

QDA (Quadratic Discriminant Analysis) [Statlog 1994]. Quadratic discrimination is similar to linear discrimination, but the boundary between two discrimination regions is now allowed to be a quadratic surface.

kNN (k-Nearest Neighbors) assigns a given vector to that class to which most of the k nearest vectors belongs, using a given distance measure.

LMDT (Linear Machine Decision Tree) [Brodley 1992] uses at each node linear discriminants and tries to reject the least important features.

IncNET (Incremental Network) [Jankowski 1999] is an ontogenic neural network, built upon the RBF architecture, which can contract and expand in the learning process optimally adjusting its size to the data structure.

MML (Minimum Message Length) [Cichosz 2000] is a decision tree algorithm, which searches for the rules in the form that requires the fewest bits, based on entropy measure.

FOIL (First Order Inductive Learning) [Cichosz 2000] is an algorithm, which uses sequential covering in searches for the rules that are no longer than required to describe the area covered by instances. Rules of the First Order Logic are more general than prepositional rules.

Naive Bayes classifier [Duda 2001] assumes that all features are conditionally independent and instead of the n-dimensional probability density function, the problem is reduced to estimation of n one-dimensional probability density functions.

SVM (support vector machines) [Vapnik 1995] searches for a hypersurface that separates vectors that belong to two different classes and keeps the maximal distance from both classes. Contrary to LDA, it separates not the original vectors but their projections in a new space.

SMART is a statistical classification and regression method, **ALLOC80** [Hermans 1982] is a discriminant analysis, **ASI** and **ASR** and **LFC** (look-ahead feature constructor) [Ster 1996] are decision trees and **RBF** is a Radial Basis Function network [Hen 2002] .

Many of the methods were used in Statlog, a large-scale European project aimed at comparison of various statistical, neural and machine learning systems for classification problems [Statlog 1994], where their more detailed descriptions can be found.

3.2. SMLP

3.2.1. Introduction

A good strategy in data mining is to extract simplest crisp logical rules first. They provide hyperrectangular decision borders in the feature space. This approximation may not be sufficient if complex decision borders are required, but it works quite well if the problem has an inherent logical structure. For many datasets crisp logical rules proved to be highly accurate, they are easy to understand by experts in a given domain, and they may expose problems with the data itself [Duch 2001].

The approach to classification and extraction of logical rules proposed here is based on the initial framework presented in [Duch 1999c]. The acronym of this approach, SMLP, may be interpreted as either "search-based MLP" or "simplified MLP". The advantages of MLP neural networks are combined with rule based systems, allowing for extraction of simple logical rules. Instead of the gradient-based methods that run into problems for discontinuous, step-like transfer functions, the training algorithm is based on search methods [Kordos 2003a, 2004b]. It leads to simplified network structures, with few connections between the hidden and output layer. Various SMLP architectures, training, and rule extraction algorithms are considered. Several sets of rules of similar accuracy can be generated, offering different advantages to domain experts.

3.2.2. SMLP Network Structure

SMLP network uses the same architecture as MLP2LN (or C-MLP2LN) network [Adamczak 2001] (chapter 3.1.2.10). However the networks use quite different training algorithms. MLP2LN uses backpropagation with variable sigmoid slopes and two adjustable regularization coefficients. SMLP can use two training algorithms: Direct Search (SMLP-DS) and Variable Step Search (SMLP-VSS). The algorithms change one weight at a time, or, if needed, SMLP-DS changes two weights at a time only in fragments of the network. SMLP-DS allows for building more diverse sets of rules (chapter 3.2.9) and if the network structure is fixed during the training, it can be frequently trained in only one training cycle, what together with the use of signal tables (chapter 2.3.2) and the step transfer functions allows for very fast training. With SMLP-VSS our control over the form of the extracted rules is not so full as in the case of SMLP-DS, but it is usually easier to apply SMLP-VSS for complex datasets.

The basic version of SMLP network is based on a 3-layer MLP architecture. Neurons implement sigmoidal or step output functions with scalar product activation (see chapter 3.2.6 for comparison). The network requires discrete input data. If the data is continuous, it must be discretized prior to the training or at the run-time by an additional network layer.

A separate input neuron is used for each discretized feature value. Thus, the number of all input neurons equals the sum of all distinct values for all features. The network input value is 1 if the feature has the value represented by a given neuron and 0 otherwise.

One hidden neuron per class is initially created. The second hidden neuron per class is added, if the results with only one neuron are not satisfactory (the indices of the neurons in Fig. 3.1. indicate the order in which the neurons were added to the network). Weights of neurons that have already been trained are frozen, minimizing calculation time and leading frequently to better results, since it corresponds to incremental learning, decomposing the task into learning general rules first and than exceptions to these rules instead of trying to modify all rules to fit the data. If the results are still unsatisfactory then the next hidden neuron is added. The number of hidden neurons per a given class should equal the number of the data clusters within this class, which cannot be joined together without decreasing the classification accuracy. Each such cluster is then represented by one disjoined rule generated by the neuron. The hidden layer performs M-of-N logic operation, which frequently can be reduced to the AND or OR operations.

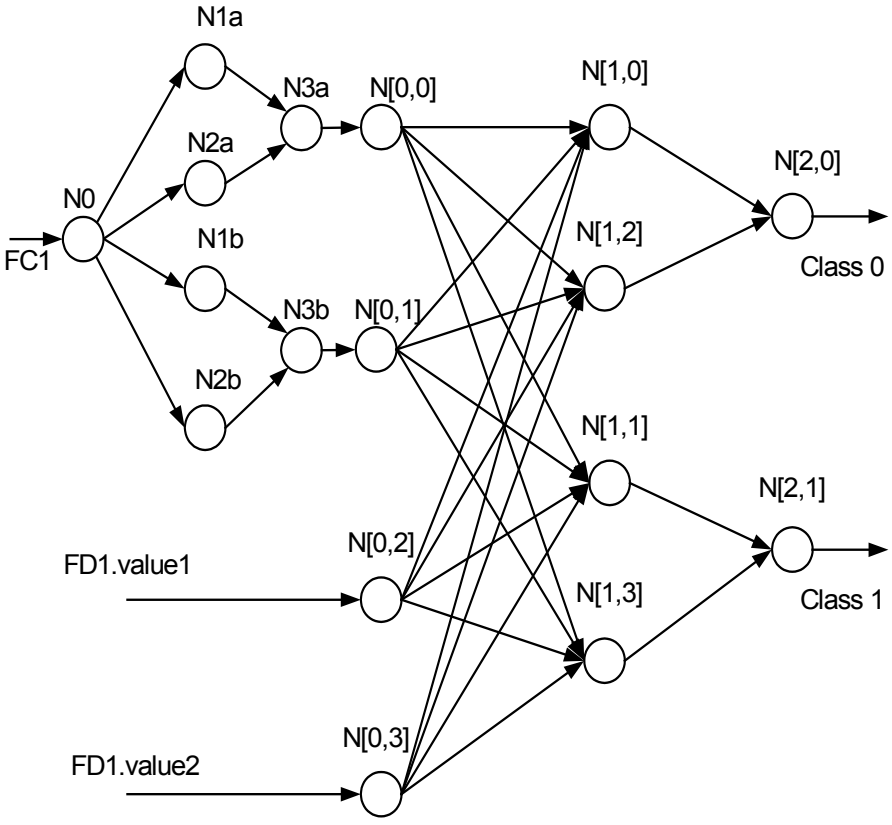

Fig. 3.1. SMLP network with a discrete feature FD1 and some pre-processing L-units for the continuous feature FC1.

There is one output neuron per class that combines the partial rules produced by hidden neurons for a given class (OR operation). The biases and weights of output neurons are constant (bias = ±0.5, weights = ±1).

The SMLP network diagram is shown in Fig. 3.1. Each value of a discrete feature (FD1.value1, FD2.value2) is given to a different input neuron. Continuous features (FC1) are discretized by logical units (L-units). There are two L-units in Fig 3.1. The first one consists

of neurons N0, N1a, N2a, N3a, and the second one of neurons N0, N1b, N2b, N3b. Signals from L-units are given to the input neurons. The L-units are discussed in chapter 3.2.8.2.

3.2.3. SMLP-DS Training Algorithms

Three SMLP-DS training methods are discussed: changing one weight at a time, changing two weights at a time and changing one weight at a time with a search strategy based on the beam search. In SMLP-DS training, the error is expressed by the standard MSE function (equation 1.3).

Only weights and biases of the hidden neurons are optimized. The weights can take only –1, 0 and +1 values if step transfer functions are used, and any integer values with sigmoidal transfer functions. The biases can take the values 0.5, 1.5, 2.5,…. up to the number of features minus 0.5. At the beginning of the training all hidden neuron weights have the value of zero and biases of 0.5. This gives no signals from the hidden to output neurons and consequently all output neuron signals are zero. Thus, at the starting point no vectors are assigned to any class.

When the training starts, the value of 1 is added to or subtracted from a single weight. If the network error decreases after the change more than the predefined threshold t (chapter 3.2.9), then the change is kept, otherwise it is rejected. The default setting is that after any error decrease the weight change is kept (t=0). Then the value of 1 is added to or subtracted from the next weight and again the error is calculated, until the changes of all weights in the hidden neuron are examined. In some cases, (e.g. for the Xor problem) changing only one parameter at a time may not be sufficient for the algorithm to converge. Thus, modifying two or more parameters at a time can be used, though it is more time consuming. Moreover, in cases where the strong asymmetry in class distribution occurs, sometimes the training may be easier and better results may be achieved using the balanced error function.

Usually one training cycle of the algorithm is sufficient as well with changing one as two weights at a time. More training cycles may be required if the threshold value t is being gradually changed.

The desired signal of each output neuron is 1 if the actual vector belongs to the class represented by this neuron, and 0 otherwise. A vector is considered to be classified correctly if the signal of the output neuron corresponding to its class is higher than signals from all other output neurons and higher than 0.5. In the case of step transfer functions, the output signals can be obviously only 0 and 1.

While determining each weight change, the error should be calculated on the whole training set. The reason for this is that in SMLP network the proper value of a given weight is determined in a single step. If the properties of the actually used training subset differ too much from the entire training set properties, then a wrong decision about the weight value may be taken. The problem is significantly easier in standard MLP networks, when the weight values are determined in many steps and therefore the semi-batch or even on-line training works well.

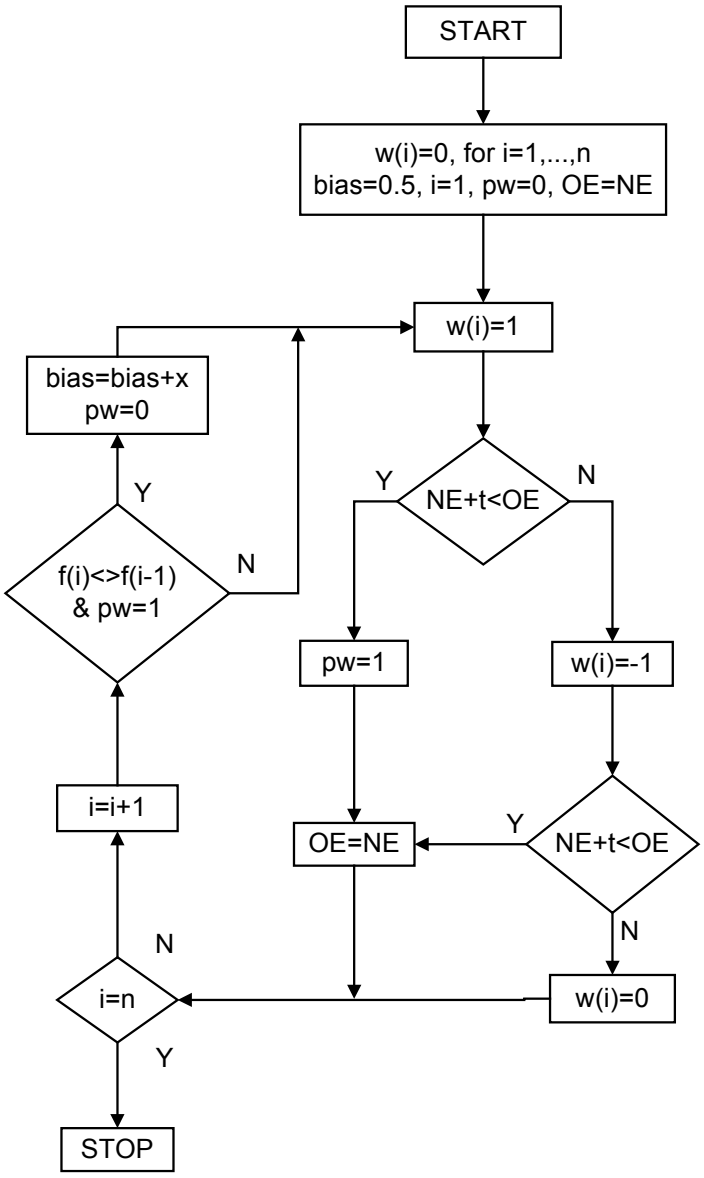

Fig. 3.2. One training cycle of SMLP-DS algorithm with changing one weight at a time.

Explanation of symbols used in Figs 3.2 and 3.3:

n – number of weights in the hidden neuron
i,j – the actually modified weight
w(i) – the i-th weight
NE – New Error, after the weight *w(i)* is changed
OE – Old Error, the lowest error value obtained (with the accepted weight change) as so far in the training
t – threshold, a given weight change is accepted if it decreases the error at least by *t*
f(i) – feature connected to the network with the weight *w(i)*
x – after at least one weight in the last group of weights connecting values of the same feature *f(i)* to the network is set to +1, *pw* is set to 1 and the bias can be left unchanged (*x*=0) or can be incremented (*x*=1), (chapter 3.2.9.2). When two weights are modified at a time, then x can be 0, 1 or 2

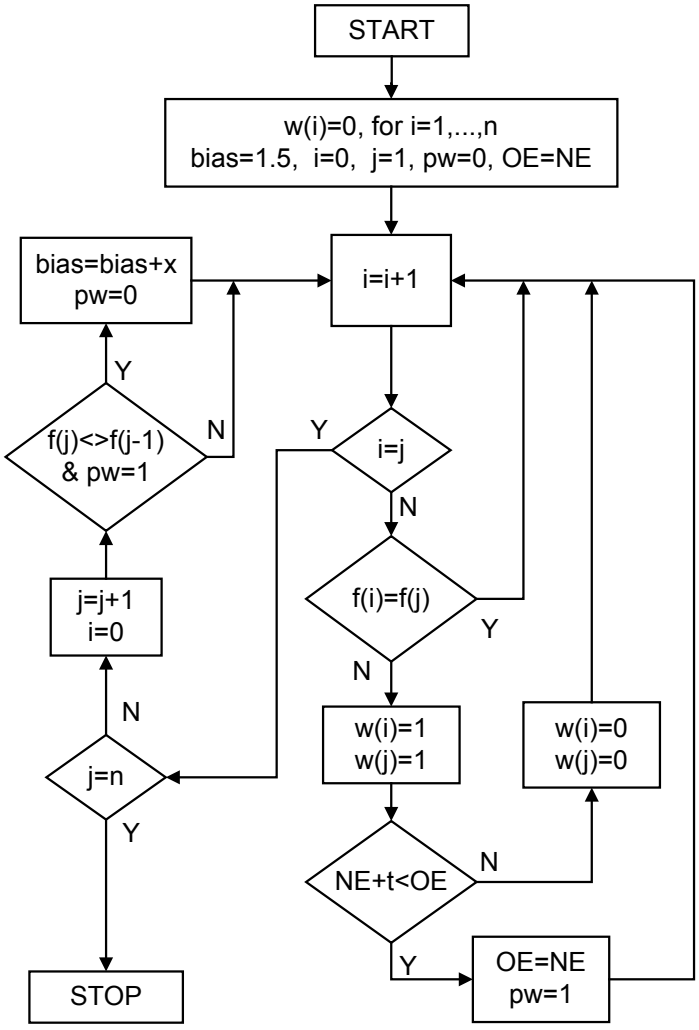

Fig. 3.3. One training cycle of SMLP-DS algorithm with changing two weights at a time.

While changing two weights at a time, there is no need to examine all possible combinations of the weight values (-1,0,+1), since all combinations with 0 on any position have already been tested while changing one weight at a time. Only the weight combination (+1,+1) is checked with bias=1.5. The weight combinations (-1,+1) and (+1,-1) can be checked only with bias=0.5 (assuming that other weight values are zero). However the lack of one feature value is equivalent to the presence of some other values and therefore it is usually enough to check the (+1,+1) weight combination. The influence of negative weights (-1,-1) can be easily examined in the "negative logic", when by default all data is assigned to a given class and only the exceptions must be found. That can be sometimes simpler. Also these two approaches can be combined in one SMLP network. However, in the majority of cases the positive logic is easier to use and better reflects natural human reasoning.

While changing one weight at a time, C_1 operations are required to examine all the possible weight changes (-1 and +1) for all N weights of the hidden neuron:

$$C_1=2N \tag{3.12}$$

The cost of changing two weights at a time is significantly higher. While changing two weights at a time, only the weight combination (+1,+1) is checked. Thus, only

$$C_2=0.5\cdot\alpha\cdot N\cdot(N-1),\ where\ \alpha\approx((f-1)/f)^2 \tag{3.13}$$

operations are required, where f is the number of features and α is a factor depending on the number of features and their values. α expresses the fact that we do not try to change simultaneously two weights of the same feature. If we assume, that different features have similar number of possible values, then approximately $\alpha\approx((f-1)/f)^2$.

Also a search strategy based on the beam search can be used. Beam search is a method based on the breadth first search (chapter 2.2) and thus the method based on beam search usually allows for obtaining very short rules without the need to search through the entire solution space. The search strategy in this method is shown in Fig. 3.4. The rules obtained with this method frequently do not have higher accuracy on the training set than rules obtained with other SMLP-DS training algorithms. Instead, they can have the simplest form with a given accuracy. That in turn allows for achieving the highest accuracy on the test set. The method can be used either at the weight level or at the feature level. Therefore, each node in Fig. 3.4. can represent as well a single weight as a single feature (a group of weights corresponding to all the values of a given feature that are determined using the simplest SMLP-DS method with changing one weight at a time). Each weight (or feature) can be a "parent node" or "child node" to each other weight (or feature). For that reason particular nodes can be visited several times during the network training.

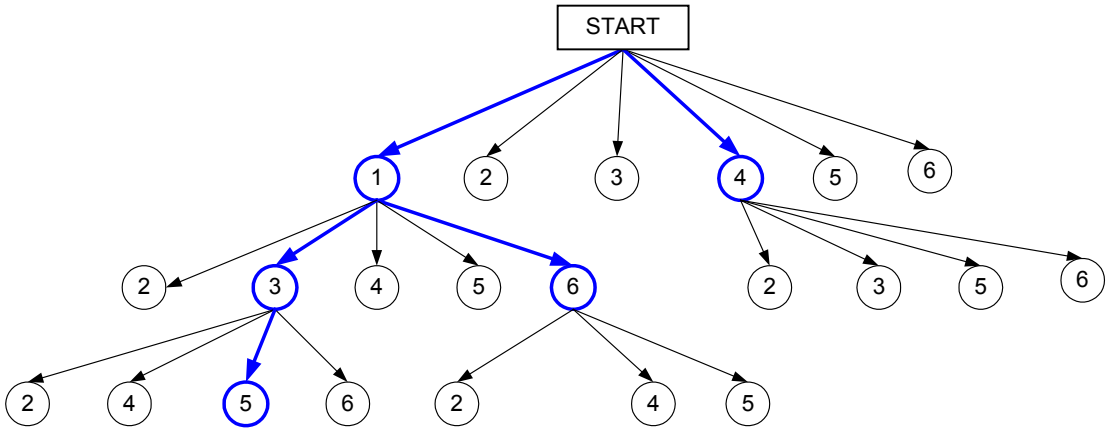

Fig. 3.4. A search strategy based on the beam search method with beam width B=2. The solution contains nodes number 1, 3 and 5. Each node can represent either a network weight or a feature, depending on the approach used.

The total number of possible rules C_T using R out of N nodes is given be the following equation:

$$C_T=\binom{N}{R}=\frac{N(N-1)...(N-R+1)}{R!} \tag{3.14}$$

The number of times a combination of nodes is calculated by the method based on the beam search C_{BS} is given by the following equation:

$$C_{BS} = N + \sum_{r=0}^{R} \sum_{b=1}^{B} (N - b - r) \tag{3.15}$$

where B is the beam width. Let N be 100, R be 5 and B be 5 then C_T=7.5·10^7 and C_{BS}=2·10^3. Let N be 100, R be 13 and B be 13 then C_T=7.5·10^{13} and C_{BS}=1.5·10^4. Thus, although this method requires much search, it is still a small fraction of the entire search space.

A good method to increase training speed is to use the signal table that has identical construction and functionality as the signal table used with NG and VSS (chapter 2.3.2). The effectiveness of the signal table used with SMLP is even higher than with NG and VSS, especially when step transfer functions are used. There are two reasons for that: first, only a single layer of weights is optimized and second, the SMLP networks operating on discretized features have more inputs than the standard MLP network (one input per each feature value, versus one input per feature in the standard MLP network), and the effectiveness of the signal table grows with the number of weights.

Table 3.1. Number of operations with and without a signal table required for one training cycle of SMLP-DS with changing one weight at a time. Ni, Nh, No – number of input, hidden and output neurons respectively.

type of operation	without signal table		with signal table
	training the entire network at once	training the network neuron by neuron	
adding incoming signals	$2[Nh(Ni+1)]^2 + Nh^2$	$2Nh(Ni+1)^2$	$2Nh(Ni+1)$
calculating neuron signals	$2Nh(Nh+No)(Ni+1)$	$2Nh(Ni+1)$	$2Nh(Ni+1)$
total number of operations	$2\{[Nh(Ni+1)]^2 + Nh^2 + Nh(Nh+No)(Ni+1)\}$	$2Nh[Ni^2+3Ni+2]$	$4Nh(Ni+1)$

Table 3.2. Number of operations with and without a signal table required for one training cycle of SMLP-DS with changing one weight at a time for the network structure 125-8-2 (Ni=125, Nh=8, No=2).

type of operation	without signal table		with signal table
	training the entire network at once	training the network neuron by neuron	
adding incoming signals	2032192	254016	2016
calculating neuron signals	20160	2016	2016
total number of operations	2052352 (100%)	256032 (12.5%)	4032 (0.20%)

With step transfer functions, we can assume that adding one incoming signal requires the same calculation time as calculating one neuron signal and therefore the operations in Table 3.1 and 3.2 are summed together. The values in Table 3.1 and 3.2 are given for one training vector with step transfer functions used by SMLP-DS algorithm. If the weights are determined using more vectors in the training set, and usually they are, the values must be multiplied by the number of training vectors. SMLP-VSS algorithm uses sigmoidal transfer

functions and in this case the efficiency of signal tables decreases about 8-25% (greater decrease for smaller networks).

Several other aspects of SMLP-DS with changing one or two weights at a time will be successively discussed in later chapters.

3.2.4. Rule Extraction

Rules are extracted after the network is trained. Therefore, the rule extraction process is exactly the same for SMLP networks trained with SMLP-DS and with SMLP-VSS algorithm.

If a given value occurs in any vector, than it is always represented by the input signal, which equals one. For all values, which do not exist in a given vector, the incoming signals are zero. If a presence of a given value contributes to a given class (*odor=A,L,N*), the hidden neuron weight will be positive. If the absence (*color=R*) - then the weight will be negative. If the value is irrelevant to this class (*color=B, odor=F*) then the weight should be zero.

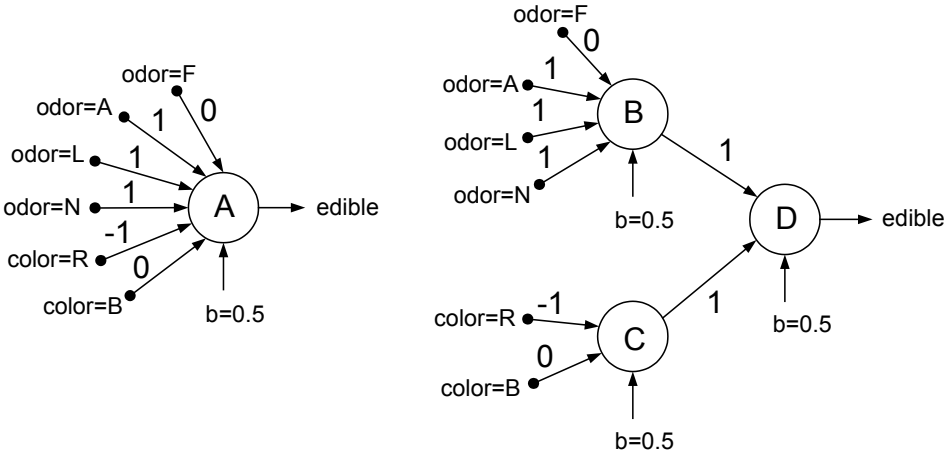

Fig. 3.5. Rules produced by hidden neurons.

In general, the hidden neurons generate M-of-N rules (if M assumptions out of N are satisfied then the condition is true). If the sum of all N inputs of a hidden neuron exceeds its bias, which has the value of M-0.5, then a logical rule is generated. Either the M-of-N rules or the AND/OR rules may describe a given problem more adequately and may be preferred in a given situation. When at least one of the N assumptions must be satisfied then the M-of-N operation can be reduced to an OR operation. When all N assumptions must be satisfied then the M-of-N operation can be reduced to an AND operation. However, in practice it cannot be done without considering the meaning of particular weights. An example is shown in Fig. 3.5. The rule generated by the network is:

if (odor=A or odor=L or odor=N) and (not color=R) then edible

The rule is represented by the three neurons in Fig. 3.5-right. When the sum of the signals incoming to neuron D (we assume that all neurons have step transfer functions) exceeds its bias, then the rule is satisfied. However, a single feature can take only one value for a given vector (for example *odor* cannot simultaneously take the value *A* and *L*) and therefore neuron B generates the OR rule. One can imagine that every time when several values of the same feature are connected to one hidden neuron, first the values are grouped together in the "OR" neurons (B, C), that are connected to the hidden neuron (D) only via single weights. The left side of Fig. 3.5. shows a diagram in which the network of neurons B, C, D is represented by a single neuron A. The simplified diagram is in fact implemented in the SMLP network. The simplified version can be used because only one value of a given feature can occur simultaneously. Values within one feature are first joined with OR operations and then the resultant feature values with M-of-N operations (which we try to reduce to OR or AND operations whenever it simplifies the rules).

The output layer performs always OR operations, combining rule conditions into final rules. This structure leads to very straightforward and comprehensive crisp logical rules that are extracted from the data by the analysis of the weights in the trained network, as shown in Fig. 3.6. using the Xor example.

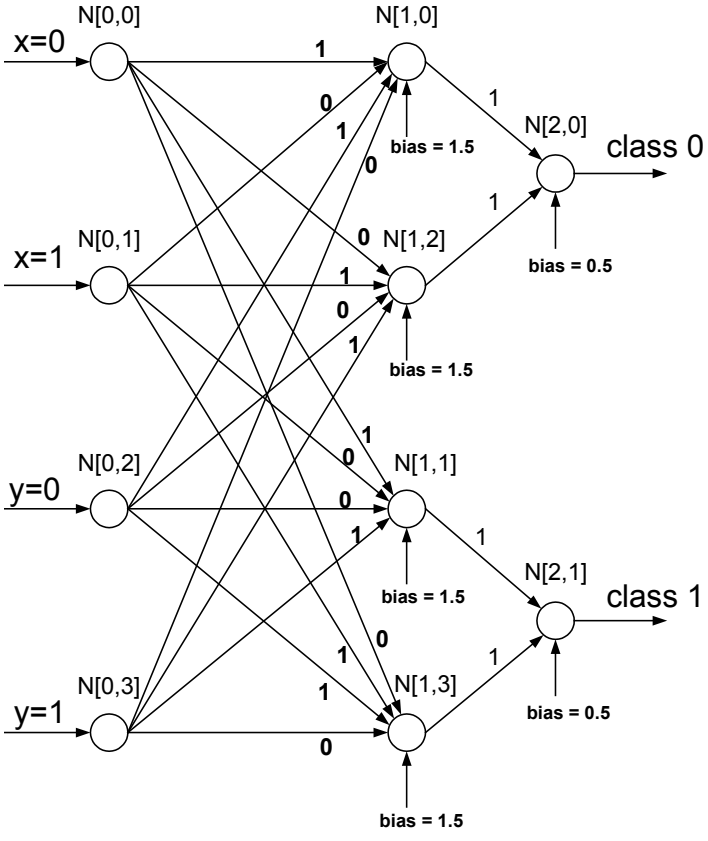

Fig. 3.6. SMLP network trained on the Xor problem.

Partial rules given by the hidden layer:

N[1,0]: *if x=0 and y=0 then class 0*
N[1,2]: *if x=1 and y=1 then class 0*
N[1,1]: *if x=0 and y=1 then class 1*
N[1,3]: *if x=1 and y=0 then class 1*

Final rules given by the output layer:

N[2,0]: if N[1,0] or N[1,2] then class 0 <=> *if (x=0 and y=0) or (x=1 and y=1) then class 0*
N[2,1]: if N[1,1] or N[1,3] then class 1 <=> *if (x=0 and y=1) or (x=1 and y=0) then class 1*

All zero weights in Fig. 3.6 could also take the value of –1, which in this case results in exactly the same rules.

The biases of output neurons can be either -0.5 or +0.5. If the output neuron bias is 0.5, then without any incoming signals, the signal of the neuron equals zero. Thus, no data is by default assigned to the class represented by the neuron. In this situation, the rules tend to be built mostly in the positive form, because some conditions must occur to activate the neuron. This configuration will be typically used in SMLP networks. However, if the output neuron bias is -0.5, then without any incoming signals, the signal of the neuron equals one. Thus, all data is by default assigned to the class represented by this neuron. In such a situation, the rules tend to be built mostly in the negative form, because some conditions must not occur to deactivate the neuron.

Examples of rules extracted from the discretized (chapter 3.2.8) Iris dataset:

Rules obtained with output neuron biases = 0.5:
if petal-length<2.5 then Iris-Setosa
if 2.5<petal-length<4.9 and not 1.7<petal-width then Iris-Virginica
if 4.9<petal-length and 1.7<petal-width then Iris-Versicolor

Rules obtained with output neuron biases = -0.5:
if not petal-length>2.5 and not petal-length>4.9 then Iris-Setosa
if not petal-length<2.5 and not petal-length>4.9 and not petal-width>1.7 then Iris-Virginica
if not petal-length<2.5 and not 2.5<petal-length<4.9 and petal-width>1.7 then Iris-Versicolor

Both sets of rules classify correctly 98% of instances. To simplify the set of rules, the rules for one of the classes can be given by “else”.

Positive conditions lead usually to simpler rules, therefore the search strategy should be arranged in such a way that negative weights in output neurons do not occur frequently.

Table 3.3. Number and accuracy of rules for the Iris dataset obtained with various rule extraction algorithms.

method	number of rules/prepositions/features	accuracy	source
FuNN	9/26/4	95.7	[Kasabov 1996]
FuNN	14/28/4	95.7	[Kasabov 1996]
NefClass	7/28/4	96.7	[Nauck 1996]
NefClass	3/2/6	96.7	[Nauck 1996]
FuNe-I	7/-/3	96.0	[Halgamuge 1994]
C-MLP2LN	2/2/2	96.0	[UMK-KMK]
C-MLP2LN	2/3/2	98.0	[UMK-KMK]
SSV	2/3/2	98.0	[UMK-KMK]
SMLP	2/2/1	95.7	this work
SMLP	2/3/2	98.0	this work

3.2.5. SMLP-VSS Training Algorithm

Sigmoidal transfer functions allow the network to use continuous error measures and therefore eliminate the need for more complex search techniques (chapter 3.2.9 and 3.2.10). The SMLP network can be modified so that it could use sigmoidal transfer functions and VSS as the training algorithm. VSS (Variable Step Search Algorithm) was introduced and described in detail in chapter 2.4.2 as a training method for standard MLP networks. VSS can be easily adapted to SMLP networks training by modifying the default parameters to adjust them to smaller weight values. The values presented in table 3.4. were empirically determined on several datasets including those presented in chapter 3.2.12.

Table 3.4. Default VSS parameters with sigmoid slope=1 for standard MLP and SMLP networks (see chapter 2.4.2 for the parameters explanation).

parameter	default values for standard MLP	default values for SMLP network
d0	0.2	0.1
d1	0.03	0.01
c1	0.33	0.22
c2	2.0	1.5
max_n	4	4
c3	0.3	0
max_w	-	1.5

The SMLP network trained with VSS algorithm uses sigmoidal transfer functions as well in the hidden as in output layer. At the beginning there is only one hidden neuron per class, more hidden neurons will be added if it is required. Only the weights of the hidden layer are optimized. At the beginning all the weights have the same values as at the beginning of SMLP-DS training. The weights of the hidden neurons are zero and biases are 0.5, the first weight of the output neuron is +1 and its bias is +0.5. If the further hidden neurons are added,

the weights connecting them to the output neurons will be either +1 (to classify the unclassified actual class vectors) or –1 (to classify the exceptions, i.e. vectors from other classes that were wrongly classified by the first hidden neuron as the actual class instances).

The standard MSE function is used, however with an additional penalty term to enforce all the weights *w* of the hidden neurons to take the values –1, 0 or +1.

$$E = \sum_{v=1}^{Nv}(out_v - desired_v)^2 + c\sum_{i=0}^{Nw}|\, w_i(w_i - 1)(w_i + 1)\,| \tag{3.16}$$

where *c* is a constant, usually $c=0.02 \div 0.08 \cdot N_v/N_w$ is an optimal value (N_v is the number of training vectors N_w is the number of weights of the hidden neuron). The VSS algorithm starts with the first guess of every weight change $dw=0.1$ and then it runs according to the diagram shown in Fig. 2.22. At the beginning all the weights are in the basin of attraction $w=0$ of the penalty term. If changing a given weight does not decrease the error than the weight stays at zero. The constant *c* can be used to regularize the complexity of the rules. With greater *c* fewer weights leave the zero basin of attraction. Changing the weights value above +1 or below -1 does not cause such significant error reduction as changing it from zero to one, first because the regularization term is stronger in these areas and second, because the sigmoid is not so steep as it is close to zero. In practice, the weights very rarely leave the +1 or –1 basin of attraction.

The regularization term is not added to the bias. This ensures that the bias can freely take an optimal value. Only in the last training cycle (for many datasets four training cycles are sufficient) a regularization term, similar to that added to weights, is added to the bias to enforce it to move to the nearest *n-0.5* value, where *n* is any integer from one to the number of features.

After the last training cycle the weights have usually the values –1±0.05, 0±0.05 and +1±0.05 and bias has the value n-0.5±0.05. Then the weights are transformed in the following way

if $w<-0.5$ *then* $w=-1$ (3.17)
if $w>0.5$ *then* $w=+1$
else $w=0$

if $n-1<bias<=n$ *then* $bias=n-0.5$, *for* $n=1$ *to number of features*

While using the network on the test set, either the original sigmoidal neural transfer functions can be used, or the sigmoid slopes can be increased, or the transfer functions can be transposed to step functions thus converting the network to the SMLP-DS form. It can be sometimes observed that after changing the transfer functions from sigmoids to step-like, the number of misclassified vectors slightly changes. There is no clear rule if leaving the sigmoidal transfer functions leads to better generalization on the test set. Sometimes it is so, probably because the decision borders with sigmoidal transfer functions do not have to be hyperrectangular and can better fit the data. But on the other hand sometimes just the hyperrectangular decision borders may be required. They can also reduce the noise in data. The best solution is to perform crossvalidation tests with the actual sigmoid sloped, with the increased sigmoid slopes and with the step transfer functions and then to decide which functions will lead to the best results.

The rules are extracted from the weight values, as discussed in chapter 3.2.4. However, if the transfer functions are not transposed to step functions than the rules extracted from the network will only approximately fit the mapping that the network performs. (Fuzzy rules with sigmoidal membership functions may be more faithful to the original network mapping.)

VSS performs better for SMLP networks than gradient-based algorithms for two reasons. First, because it does not change all the weights at once, but weight by weight and if changing the previous weight already significantly reduces the error, than the next weight is not able to leave the zero basin of attraction. Thus, the network and the rules are kept simple. The complexity of the rules can be tuned by the regularization constant *c*. And second, VSS uses individual steps for each weight, which allows the weights to take optimal values very quickly.

The possible risk of SMLP training with VSS is that after adding the second neuron, its weights may take the same values as the first neuron weights. (In SMLP trained with step transfer functions the risk does not exist.) The vectors of the actual class that were correctly classified by the first hidden neuron can be removed from further training only in this case, when the first neuron did not classify any other classes vectors as the actual class vectors (chapter 3.2.9.2). The regularization term in the error function cannot be increased to prevent redundant neuron roles, because in this case it would also prevent other weights from leaving the zero basin of attraction. Thus, the first term of the error function must be modified in such a way that if a given vector is already classified correctly by the first neuron than the correct classification of it by the next neuron will not change the error:

$$\textit{for } n=2 \textit{ to } N \textit{ do} \qquad (3.18)$$
$$\textit{if } Accuracy(v, n-1)=100\% \textit{ than } Error(v, n)=\max(Error(v, n-1), Error(v, n))$$

where n is the current hidden neuron, N is the number of hidden neurons per given class, v is the vector number, $Error(v, n)$ is the MSE error generated in response to vector v by the network with n hidden neurons. This ensures that the next neuron will not perform the same functions as the previous one.

3.2.6. Step Versus Sigmoidal Transfer Functions

In most cases step transfer functions are used with SMLP-DS training. In comparison with sigmoidal functions, they produce simpler rules usually of comparable accuracy. Step functions give only the information that is necessary to classify a vector. With step functions when a vector is classified correctly the error already equals zero and no additional input conditions can decrease it, so the conditions do not come into the final rule.

Sigmoidal functions, in addition to the information needed for classification, provide also information about other feature values, specific to a given class but not required by the classification process. With sigmoidal functions (without weight regularization), adding more conditions to a rule may still decrease the network error, since the output signal is always lower than 1 and always can be increased. Moreover, the number of additional conditions of the rule may be regulated by the required output accuracy, assuming that output values above

some threshold (e.g. 0.98) are considered as 1. For example, the rule for one class of the Iris dataset obtained using step transfer functions is:

if petal-length<2.5 then Iris-Setosa

The rule obtained with sigmoidal transfer functions has two conditions:

if petal-length<2.5 and petal-width<0.8 Iris-Setosa

The second condition is not necessary for classification, but provides additional information about the data properties. The network can also be trained using step transfer functions to extract logical rules, then the functions can be changed to sigmoids, and an additional SMLP-DS training cycle can be run to provide some extra information about the data. Moreover, if the dataset is small and has many features a single rule condition may work well due to accidental distribution of training data, so it may be better to use additional conditions.

The optimal cut-off points determine a hypersurface that separates vectors that belong to two different classes and keeps the maximal distance (maximal margin) from both classes. The points are in this case about 2.5 for *petal-length* and about 0.8 for *petal-width* (Fig. 1.14-left-top). That ensures the highest test accuracy and stability of the classifier.

3.2.7. Feature Selection

In datasets that contain many attributes, usually only some of them provide useful information. Using all the attributes for the training of SMLP networks causes two problems. First, the training time is unnecessarily long. Second, while changing one or two parameters at a time with SMLP-DS, the order in which the weights are examined plays a role. If uncontrolled, this effect can adversely influence the training, because the extracted rules depend on the training process in an unforeseen way. On the other hand if the order is controlled it can provide us with various sets of rules, thus enhancing our knowledge about the dataset more than a single best optimized set of rules. However, with many attributes, it gets difficult to extract the proper rules. This problem is dealt with using feature selection based on the information included either in the single feature, or in a set of features.

The usefulness of various features for classification purposes can be assessed by feature ranking and feature selection. In feature ranking, the predictive abilities (the classification accuracy obtained by using only this single feature) of each feature are assessed and then the features are decreasingly ordered according to that assessment. Ranking is the simplest method, however it does not always work well, because the independent assessment of each feature is not always related to the assessment of the group of features. For example, the second feature in the ranking may carry practically the same information as the first one. Then the space of the two features gives the same classification accuracy as the first feature alone. At the same time, a feature that has a further, e.g. fourth, position in the ranking can be a useful source of information, efficiently completing the information contained in the first feature. Thus, the first and fourth feature may be the best choice and not the first and the second one.

Feature selection methods assess various combinations of features, usually using some iterative algorithms. Feature selection is computationally more costly than feature ranking but can bring better results. Generally, the feature selectors can be divided into filters and wrappers. Filters assess the usefulness of particular features independently of the classification algorithm that will be used with the selected features. Wrappers cooperate with particular classifiers and usually require multiply runs of the classification algorithm. Thus, wrappers may be very costly and for that reasons frequently the results obtained with filters are more effective.

To assess the mutual information between the values of a given feature and the class of instances, the feature must be discrete. If the feature is continuous, it must be discretized (chapter 3.2.8) prior to using the filter.

Information contained in the joint distribution of classes and features, summed over all classes, gives an estimation of an importance of the feature:

$$I(C,X) = -\sum_{i=1}^{K} \int p(C_i, X=x) \lg_2 p(C_i, X=x) dx \approx -\sum_{x} p(X=x) \sum_{i=1}^{K} p(C_i, X=x) \lg_2 p(C_i, X=x) \tag{3.19}$$

where $p(C_i,X=x),\ i=1...K$ is the joint probability of finding the feature value $X=x$ for vectors $\boldsymbol{X}$ that belong to some class C_k (for discretized continuous features $X=x$ means that the value of feature X is in the interval x) and $p(X=x)$ is the probability of finding vectors with the feature value $X=x$, or within the interval $x \in X$. Low values $I(C,X)$ indicate that vectors from a single class dominate in some intervals, making the feature more valuable for prediction [Duch 2003].

Joint information may also be calculated for each discrete value of X or each interval and weighted by the p(X=x) probability:

$$WI(C,X) = -\sum_{x} p(X=x) I(C, X=x) \tag{3.20}$$

Information contained in the $p(X=x)$ probability distribution plus the $p(C)$ class distribution minus the joint information $I(C,X)$ is called "mutual information" or "information gain"

$$M_I(C,X) = I(C) + I(X) - I(C,X) \tag{3.21}$$

Mutual information is equal to the expected value of the ratio of the joint to the product probability distribution, known as Kullback-Leibler divergence:

$$M_I(C,X) = E\left(\frac{p(C,X)}{p(C)p(X)} \right) = D_{KL}\left(p(C,X) \,\|\, p(C)p(X) \right) \tag{3.22}$$

A feature is more important if its mutual information is larger.

Several approaches to feature selections can be used with SMLP networks:

- use the information provided by some external filters
- first perform the feature ranking, based on the information contained in a single feature and then add the features in the order in which they appear in the ranking
- add gradually features to the rule using a method based on the beam search
- use all the features with the threshold *t* added to the error function. If including a given feature does not decrease the error more than the threshold value, the feature is not added at this moment. However, the threshold is being gradually decreased during the training, so the feature will have a chance to be included further in more specific rules, which cover fewer instances.

3.2.8. Feature Discretization

There are two objectives while discretizing continuous data: to have a few discrete values in order to obtain a simple network and simple rules, and to have enough discrete values for accurate rules and reliable classification results.

3.2.8.1. Prior to Training Discretization

There are many discretization methods [Liu 2002], which divide the continuous data into intervals basing on various criteria for setting the split points. However, so far only equal width and equal frequency discretization was used with SMLP. Then the adjacent intervals were merged and the split points of the most important features were fine-tuned.

Initially each continuous feature space is divided into *n* equal width or equal frequency intervals (n=10 is sufficient in most cases). The merging of adjacent intervals can be done in two ways: on-line during the training and before the training basing on the idea of 1R quantization method [Holte 1993]. The on-line method is in general more accurate, because it considers also interaction between various features. However, frequently it is beneficial to reduce the number of intervals before the training.

Holte avoids large number of intervals by requiring all intervals (except the rightmost) to contain more than a predefined number of examples in the same class. Empirical evidence led him to a value of six for datasets with more than 50 instances and three for smaller datasets. Each interval is assigned to the class to which the majority of its vectors belong. Then adjacent intervals assigned to the same class are joined.

3.2.8.2. Run-time L-unit Based Discretization

The initial interval boundaries obtained from the prior-to-training discretization may be tuned with L-units (logical units) using search techniques. An interval cut-off point in the most significant feature is shifted and the training is performed. If the error decreases then the shift was in a proper direction, otherwise it was in the wrong direction. The procedure may be repeated with each interval cut-off point for all features. Features that are useless for discrimination of a given class (shifting the cut-off points does not influence training results) are automatically removed.

Continuous features are given to the inputs of L-units. Each L-unit passes only these values, which are within its discretization window, thus as many L-units per feature are needed as the required number of that feature discrete values. Since discretization and learning are done in the same network, results depend on the whole training set, not just on the single feature being discretized.

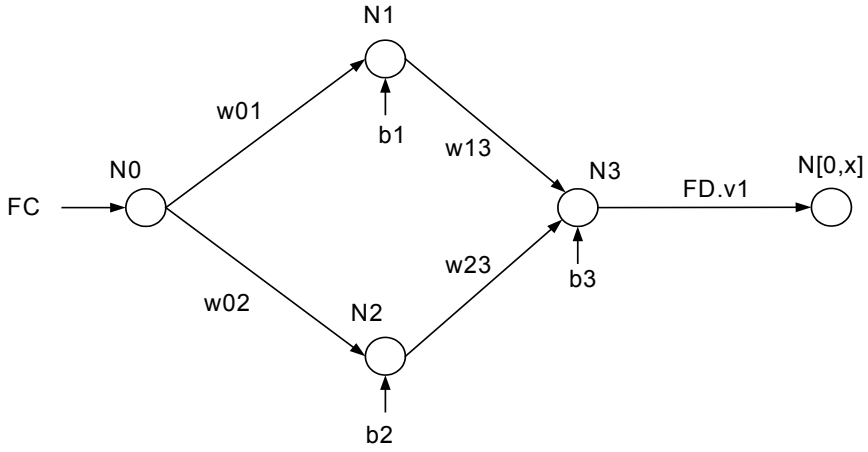

Fig. 3.7. L-unit (neurons N1,N2,N3) with the feature input (N0) and the SMLP network input neuron (N[0,x]).

The continuous feature *FC* given to the neuron N0 (which has linear transfer function and is used only to distribute the feature value to further neurons) is connected with weights w_{01} and w_{02} to the neurons N1 and N2. A single N0 neuron is used by all L-units assigned to the same continuous feature. The weights w_{01} and w_{02} have constant values set to +1. The weights are not modified during the learning process. Neurons N1, N2 and N3 have step transfer functions. The output signals of N1 and N2 can take values -1 or 1. The output signal of N3 can take values 0 or +1 and it is passed to the corresponding input neuron of the SMLP network (neuron N[0,x]). The L-unit realizes the following function:

$$FD.v1=0.5+0.5\cdot sign(w_{13}\cdot sign(FC-b_1)+w_{23}\cdot sign(FC-b_2)-b_3) \tag{3.23}$$

Neuron N1 is activated if $FC>b_1$. Neuron N2 is activated if $FC>b_2$. If neurons N1 and N2 are activated then their output signal is +1 otherwise it is –1. The biases b_1 and b_2 are modified during network learning to optimally tune the cut-off points. Neuron N3 can realize any superposition of the signals from neurons N1 and N2, depending on the weights w_{13} and w_{23}, which can take the values –1 or +1 and on the bias b_3 value, as shown in Fig. 3.8.

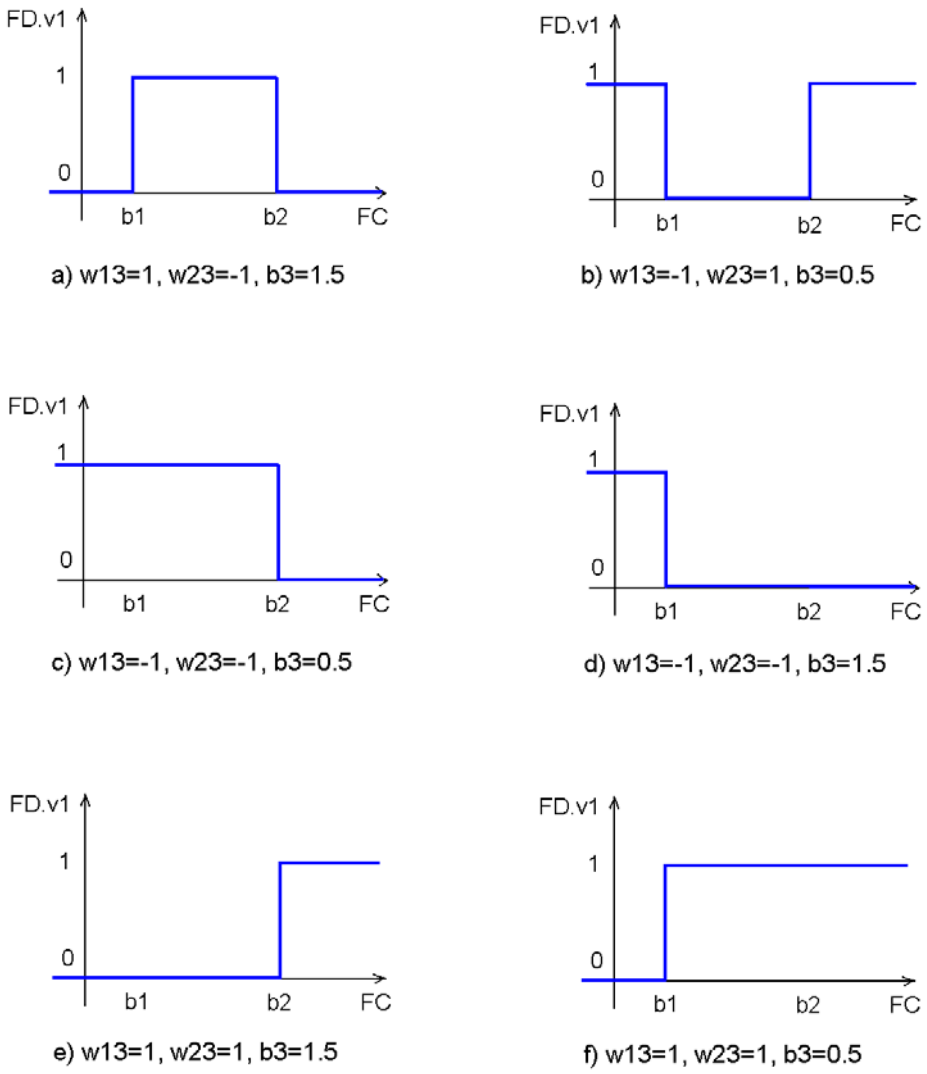

Fig. 3.8. Functions realized by L-units.

3.2.9. Advanced SMLP-DS Training Methodology

3.2.9.1. The Training Algorithm

The SMLP-DS and SMLP-VSS algorithms are the basis of the SMLP training process. The SMLP-DS algorithm will be used in the following training methodology:

1. Discretize continuous features.

2. Sort features according to feature ranking or feature filter. While building the feature ranking it may be required to tune precisely the discretization cut-off points using L-units. Frequently low-ranking features can be rejected.
3. Build the network with one hidden neuron for a given class. If the class distribution is highly asymmetric or there are more than two classes then it may be required to assign appropriate weights to the classification errors made on different class instances (to use the balanced accuracy).
4. Train the neuron changing one weight at a time starting from changing the weights of the most important feature in the ranking. Tune precisely the discretization cut-off points using L-units if necessary.
5. If the error does not decrease significantly then change two weights at a time. Tune precisely the discretization cut-off points using L-units if necessary. If changing two weights at a time does not work (and the network is still far from overfitting), three weights can be changed, SMLP-VSS or genetic algorithms used, but it is most likely that there is a problem with the consistency and reliability of the dataset itself.
6. If the error no longer decreases or the number of rule prepositions grows rapidly then freeze the weights of this neuron, add the next hidden neuron and repeat points 4 and 5. If adding the next neuron does not change the situation then go to point 7.
7. If there are only two classes, then stop the training; the rules for the second class will be given by "else". If there are three classes, repeat steps 3-6 with the second class, the rules for the third class will be given by "else". If there are more than three classes, then repeat steps 3-6 with every class, the "else" rule may be too difficult for interpretation in this case.

3.2.9.2. Sample SMLP Training on the Mushrooms Dataset

The methodology will be discussed on the Mushrooms dataset example. The dataset was constructed basing on mushroom records drawn from "The Audubon Society Field Guide to North American Mushrooms" by G. H. Lincoff. This dataset includes descriptions of samples corresponding to 23 species of mushrooms. Each species is identified as edible or poisonous one. The guide clearly states that there is no simple rule for determining the edibility of a mushroom. The dataset contains 8124 vectors, 4208 (51.8%) of them in the first "edible" class and 3916 (48.2%) in the second "poisonous" class. The dataset contains the following values of the 22 discrete features:

f1: cap-shape (bell=B, conical=C, convex=X, flat=F, knobbed=K, sunken=S)
f2: cap-surface (fibrous=F, grooves=G, scaly=Y, smooth=S)
f3: cap-color (brown=N, buff=B, cinnamon=C, gray=G, green=R, pink=P, purple=U, red=E, white=W, yellow=Y)
f4: bruises (bruises=T, no=F)
f5: odor (almond=A, anise=L, creosote=C, fishy=Y, foul=F, musty=M, none=N, pungent=P, spicy=S)
f6: gill-attachment (attached=A, descending=D, free=F, notched=N)
f7: gill-spacing (close=C, crowded=W, distant=D)
f8: gill-size (broad=B, narrow=N)
f9: gill-color (black=K, brown=N, buff=B, chocolate=H, gray=G, green=R, orange=O, pink=P, purple=U, red=E, white=W, yellow=Y)
f10: stalk-shape (enlarging=E, tapering=T)
f11: stalk-root (bulbous=B, club=C, cup=U, equal=E, rhizomorphs=Z, rooted=R, missing=?)

f12: stalk-surface-above-ring: (fibrous=F, scaly=Y, silky=K, smooth=S)
f13: stalk-surface-below-ring: (fibrous=F, scaly=Y, silky=K, smooth=S)
f14: stalk-color-above-ring: (brown=N, buff=B, cinnamon=C, gray=G, orange=O, pink=P, red=E, white=W, yellow=Y)
f15: stalk-color-below-ring: (brown=N, buff=B, cinnamon=B, gray=G, orange=O, pink=P, red=E, white=W, yellow=Y)
f16: veil-type: (partial=P, universal=U)
f17: veil-color: (brown=N, orange=O, white=W, yellow=Y)
f18: ring-number: (none=N, one=O, two=T)
f19: ring-type: (cobwebby=C, evanescent=E, flaring=F, large=L, none=N, pendant=P, sheathing=S, zone=Z)
f20: spore-print-color: (lack=K, brown=N, buff=B, chocolate=H, green=R, orange=O, purple=U, white=W, yellow=Y)
f21: population: (abundant=A, clustered=C, numerous=N, scattered=S, several=V, solitary=Y)
f22: habitat: (grasses=G, leaves=L, meadows=M, paths=P, urban=U, waste=W, woods=D)

The features can take together 125 different values. Thus, the SMLP network must have 125 inputs. The SMLP network can be trained for each class separately, since specific hidden neurons are dedicated to particular classes. The Mushrooms dataset contains two classes: *edible* and *poisonous*. It this case, it is sufficient to train the network for one class, the rules for the other class will be given by the negation of the rules obtained for the trained class. The network will be trained for the class *poisonous*. Examining all features in the order in which they appear in the original vectors and changing one weight at a time while training the network with one hidden neuron gave the following equation that must be satisfied for the poisonous class:

if 0.5 < cap_shape(C) + cap_surface(G) + cap_surface(S) + cap_color(B) - cap_color(G) + cap_color(P) - cap_color(Y) + bruises(F) – odor(A) - odor(L) +odor(C) + odor(F) + odor(M) - odor(N) + odor(P) - gill_attachment(A) + gill_spacing(C) - gill_spacing(W) - gill_color(K) + gill_color(R) - gill_color(O) - gill_color(E) - gill_color(Y) - stalk_root(C) - stalk_root(R) - stalk_surface_above_ring(Y) + stalk_surface_above_ring(K) - stalk_surface_below_ring(K) + stalk_color_above_ring(C) - stalk_color_above_ring(O) - stalk_color_above_ring(E) - stalk_color_below_ring(E) + stalk_color_below_ring(Y) + ring_type(E) - ring_type(F) + spore_print_color(R) + population(C) - population(N) - population(Y) - habitat(W) then poisonous

which gives 99.78% accuracy and classifies 4206 of edible mushrooms as edible, two edible as poisonous, 3900 poisonous as poisonous and 16 poisonous as edible. This can be written in the confusion matrix (chapter 3.2.12.1) form:

	E	P
E	4206	2
P	16	3900

The main problem is that it is not easy to draw any conclusions from this rule. We can only see which feature values positively contribute to that class (sign +) and which negatively (sign -), but it is very difficult to say what condition must exactly be satisfied to obtain the sum of incoming signals multiplied by their weights greater than 0.5. Because the rule is so complex, it would not be a good idea to add the second neuron to further improve the training accuracy (and complicate the rule even more).

Weights that belong to the same feature and have the same sign are called “group of weights”. If there are many negative and many positive groups of weights and the bias value is lower then the number of positive weight groups minus 0.5, as in the rule above, then the rule interpretation may be ambiguous. In general, when the bias value is M-0.5, the positive weight groups are interpreted as M-of-N rules, where N is the number of groups of positive weights. Any negative weight is interpreted as “and not”. Negative weights of some feature values can be replaced by the positive weights of other values of the same feature. In order to obtain the “and” interpretation, the bias must take a higher value, equal to the number of positive weight groups minus 0.5. The desired situation is when the rules can be clearly formulated according to the guidance above and frequently it is the case. If they cannot be, then either the interpretation “and not” for negative and “M-of-N” for positive weight groups should be used or the functionality of the neuron should be split among more neurons (what will be discussed later).

In general, the rules should be kept as clear and simple as possible. The simplest rules are usually obtained when the search through feature values is ordered according to the decreasing mutual information of the features. A good approach is to use some form of feature selection. This dataset has 22 attributes and usually if there are so many attributes, most of them are useless for classification. It is easiest to assess the information contained in each single feature. Training the network on each feature separately gave the accuracies shown in Table 3.5. Thus, automatically a very simple rule, which gives 98.52% accuracy, was obtained:

if odor=(C or Y or F or M or P or S) then poisonous

with the following confusion matrix*:*

	E	P
E	4208	0
P	120	3796

Searching first through values of a single feature is advantageous because it usually leads to the simplest and most comprehensive rules. Than the features were sorted according to the decreasing information gain and the last 10 features in the ranking were discarded. This decision was based on the observation, that in datasets with many features, frequently most of the features (the low-ranking ones) are irrelevant for classification. However, in more complex cases, rather a feature filter assessing the mutual feature information in connection with other features than a simple feature ranking is preferred. If the decision proved wrong in a given case then the training would have to be repeated including all the features.

Together with, or alternative to sorting the features, the threshold t can be used to keep the rules simple. The threshold is especially useful if the features are not sorted according to a feature filter that includes correlations among features and this is the case in the actual training, since only a simple feature ranking was used. At the beginning of the training, the network has one hidden neuron with the error threshold t being arbitrary set to 20.

Table 3.5. Information contained in single features of the Mushrooms dataset.

feature	accuracy [%]	accuracy-default [%]
odor	98.52	46.72
spore_print_color	86.80	35.00
gill_color	80.50	28.70
ring_type	77.54	25.74
stalk_surface_above_ring	77.45	25.65
stalk_surface_below_ring	76.61	24.81
gill_size	75.62	23.82
bruises	74.40	22.60
population	72.18	20.38
stalk_color_above_ring	71.64	19.84
stalk_color_below_ring	71.44	19.64
habitat	69.03	17.23
stalk_root	63.81	12.01
gill_spacing	61.59	9.79
cap_surface	59.52	7.72
cap_color	59.29	7.47
cap_shape	58.05	6.25
stalk_shape	55.29	3.49
ring_number	53.81	2.01
veil_color	51.89	0.09
gill_attachment	51.80	0
veil_type	51.80	0

After one epoch of training, changing one weight at a time some weights of the hidden neuron took non-zero values and the neuron generated the following rule:

if odor=(C or Y or F or M or P or S) or spore_print_color=R or stalk_color_below_ring=Y then poisonous

The rule gives 99.51% accuracy and the following confusion matrix:

	E	P
E	4208	0
P	24	3892

The accuracy is a bit lower than previously (99.78%) but the rule is clear. Now we must add the second hidden neuron for that class that will classify the vectors that were not classified by the first neuron. Thus, the weights of the first neuron will no longer by modified.

The vectors of the *poisonous* class that have already been correctly classified can be removed from the further training. That is because hidden neurons work in parallel and these vectors have already found their path through the first neuron, which they will use, no matter what the weights of the second neuron will be. However, the correctly classified instances of the *edible* class cannot be removed from the further training, because in this case it may

happen, that the weights of the second neuron could take such values that would let the *edible* class vectors pass through.

The training of the second neuron starts with bias=0.5 and changes one weight at a time. However, since it did not work here, two weights had to be changed at a time. It does not necessarily mean that results of similar quality could not be achieved changing only one weight at a time, examining the weights in a different order.

After the training, the second neuron generated the following rule:

if 2 of (gill_size=N, stalk_surface_above_ring=K, population=C) then poisonous

Although the neuron classifies the data correctly, giving 100% accuracy together with the first neuron, it generated the M-of-N (2-of-3) rule instead of the AND rule, as it was expected. The rule is equivalent to the following disjunctive normal form rule:

if (gill_size=N and population=C) or (gill_size=N and stalk_surface_above_ring=K) or (population=C and stalk_surface_above_ring=K) then poisonous

The rule can be decomposed into a minimal number of AND rules, by performing the training in the following way: first all pairs of weights are checked with the weights being set again to zero before the next pair is examined. Then the best pair of weights is selected and it gives the first AND rule. In this case, the rule is:

if gill_size=N and stalk_surface_above_ring=K then poisonous

The rule gives 99.90% accuracy and the following confusion matrix:

	E	P
E	4208	0
P	8	3908

Then the third hidden neuron was added and trained changing two weights at a time. It generated the following rule:

if gill_size=N and population=C then poisonous

The rule covered the remaining 8 vectors, achieving 100% accuracy on the training set. That means that the third AND rule *(population=C and stalk_surface_above_ring=K then poisonous)* covers either only the instances contained in the other rules or an empty set.

Finally the following rules were obtained:

if odor=(C or Y or F or M or P or S) (98.52%)
or spore_print_color=R or stalk_color_below_ring=Y (99.41%)
or (gill_size=N and stalk_surface_above_ring=K) (99.90%)
or (gill_size=N and population=C) then poisonous (100%)
else edible

The rules can also be written in the shorter form:

if not odor=(A or L or N) or spore_print_color=R or or stalk_color_below_ring=Y
or 2 of (gill_size=N, stalk_surface_above_ring=K, population=C) then poisonous else
edible (100%)

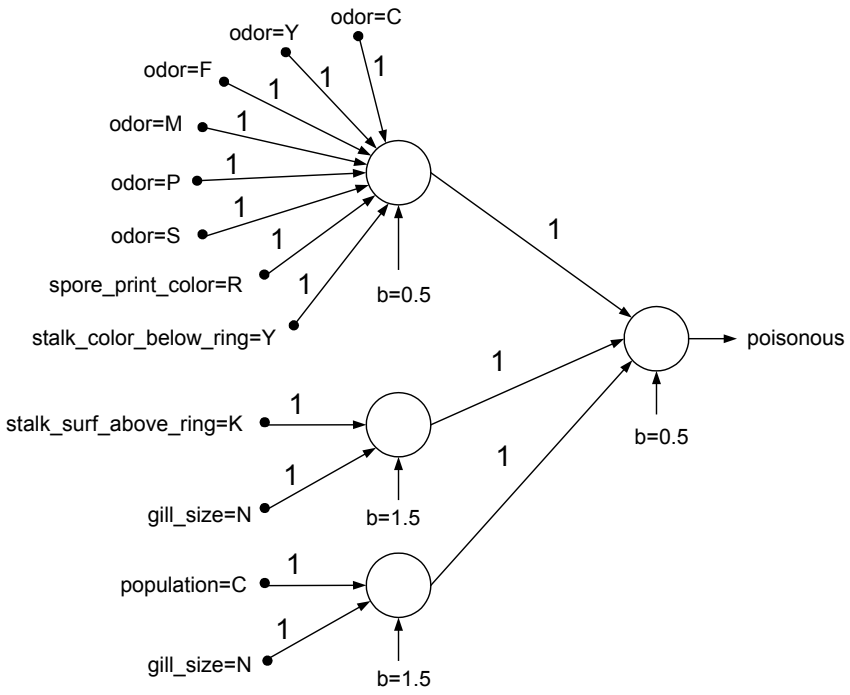

Fig. 3.9. SMLP network obtained for the *poisonous* class of the Mushrooms dataset. Only non-zero weights are shown.

Though the rules could be obtained with two hidden neurons, it seems that with three hidden neurons their form is more convenient. When at least four of these rules are used, than the third one is redundant, however it may be a compromise between the accuracy of the fourth rule and the simplicity of the second one. The redundancy was not detected by the training algorithm. It would have been detected, if the threshold *t* of the first neuron had been set to a higher value. Then, the second neuron would have been added earlier and trained changing two weights simultaneously. Changing two weights at a time sometimes allows for extracting more accurate and simpler rules, but on the other hand, the computational cost of such an approach is significantly higher. For datasets with relatively simply structure it is usually sufficient to change one weight at a time.

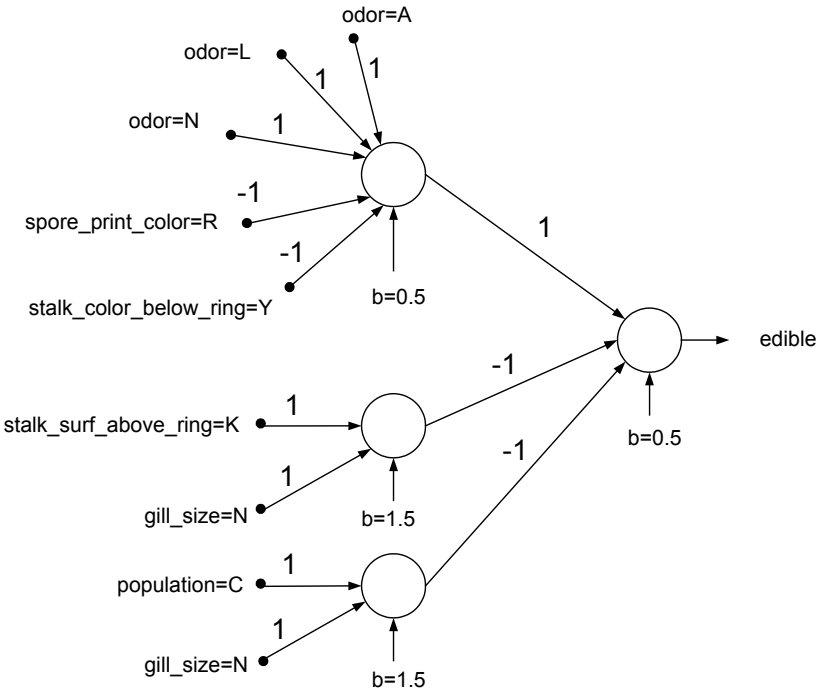

Fig. 3.10. SMLP network obtained for the *edible* class of the Mushrooms dataset. Only non-zero weights are shown.

In the training performed on the Mushrooms dataset, already the first rule was not satisfied by any *edible* mushrooms. But if the first neuron classified also some *edible* mushrooms as *poisonous*, then the next hidden neuron should be added with a negative weight (-1) to the output neuron. Then during the training, the neuron would learn to recognize the *edible* mushrooms misclassified by the first one as *poisonous*. Thus, the first neuron would provide a general rule and the second one an exception from that rule. There can be several hidden neurons with –1 weight to the output neuron to classify exceptions, as well as several hidden neurons with +1 weight to the output neuron to classify smaller clusters of the actual class instances. For example, if the training of the Mushrooms dataset starts from the *edible* class, than the first neuron generates the rule:

if odor=(A or L or N) and (not spore_print_color=R) and (not stalk_color_below_ring=Y) then edible

However, the rule covers also 24 poisonous instances. Thus, the second neuron with –1 weight to the output neuron must be added to generate the rule for the *poisonous* mushrooms and if it does not cover all *poisonous* mushrooms covered by the first neuron – than also the third neuron with –1 weight to the output neuron must be added (Fig. 3.10). Thus, the *poisonous* class vectors will activate the first neuron, but also the second and the third neuron. That will result in no activation of the output neuron and thus in correct classification.

Table 3.6. Number and accuracy of rules for the Mushrooms dataset obtained with various rule extraction algorithms.

method	number of rules/prepositions/features	accuracy	source
RULENEG	300/8087/-	91.0	[Sestito 1994]
REAL	155/6603/-	98.0	[Craven 1996b]
DEDEC	26/26/-	99.8	[Tickle 1994]
RULEX	1/3/1	98.5	[Andrews 1994]
Successive Regulariz.	1/4/2	99.4	[Duch 1997b]
Successive Regulariz.	2/22/4	99.9	[Duch 1997b]
Successive Regulariz.	3/24/6	100	[Duch 1997b]
C-MLP2LN, SSV	1/3/1	98.5	[UMK-KMK]
C-MLP2LN, SSV	2/4/2	99.4	[UMK-KMK]
C-MLP2LN, SSV	3/7/4	99.9	[UMK-KMK]
SSV	4/9/5	100	[UMK-KMK]
C-MLP2LN	4/9/6	100	[UMK-KMK]
SMLP	1/3/1	98.5	this work
SMLP	2/4/2	99.4	this work
SMLP	3/5/3	99.7	this work
SMLP	3/7/4	99.9	this work
SMLP	4/12/6	100	this work
SMLP	4/9/5	100	this work
SMLP	3/8/5	100	this work

3.2.10. Comparison of SMLP and Standard MLP Networks

Let's assume that the data has two features *F1* and *F2*. The rule can be "class 1 if *F1* and *F2"*. However, it is likely that in the areas, described by "*F1* and not *F2"* or "not *F1* and *F2"* there will be more class 1 instances than in the area described by "not *F1* and not *F2"*. Thus, looking at the error surface for this problem there exist a smooth transition, or an addition stair, that can be traversed changing only one weight at a time (additionally the bias can be incremented form 0.5 to 1.5, but this does not require significant computational effort). Changing two weights at a time with SMLP-DS is required if there is no smooth transition between two areas in the data and the error does not decrease until both weights are changed at once. Theoretically, in n-dimensional data space with no smooth transitions, there may be a need to change n weights at once, but the need to change more than two weights has not been observed so far on real-world datasets.

In SMLP trained with VSS and in standard MLP networks the problem is solved by using a continuous error measure with continuous neural transfer functions. If the training moves in the proper direction, then even if the number of correctly classified vectors does not actually grow, the error decreases. That allows for an easy training of MLP and SMLP networks with VSS algorithms, which changes only one weight at a time. The fully discretized SMLP-DS network does not provide this possibility.

Table 3.7. Comparison of MLP and SMLP networks.

	MLP	SMLP
decision borders of a single neuron	any hypersurface	hyperrectangle with sides parallel to feature axes
desicion borders of the network	any hypersurface combination with a tendency for edge smoothing	any combination of hyperrectangles with sides parallel to feature axes
required number of hidden neurons	depending on decision borders, the number can be higher either in MLP or in SMLP network	
generalization abilities	depending on decision borders, either in MLP or in SMLP network can generalize better	
information storing	globally (difficult to say what a single weight is responsible for)	locally (each weight is explicitely assigned a specific role)
rule extraction	complex and difficult	simple and easy
constructive algorithm	possible	embeded in the model
required number of training cycles	at least several	one training cycle is frequently enough, though more cycles can lead to more efficiant rule sets
weight pruning	possible	unneeded (excessive connections are not created)

The decision borders of SMLP network are hyperrectangular in the discretized search space. Nevertheless, they do not have to be hyperrectangular in the original continuous search space. For example, a new feature that is the sum or product of the some most important continuous features can be created and its value first calculated in the original continuous space and then discretized. Pao examined networks with additional inputs (called by him functional link nets) [Pao 1989] of several kinds and found that the combination of some inputs were frequently very useful.

Another method leading to decision borders, which are not parallel to the feature axes in the original continuous space is PCA (Principal Component Analysis). PCA produces new features that are weighted sums of the original features and that can be used in the feature space in the same way, as PCA directions were used in the MLP weight space (chapter 1.2.3.2). Additional functional inputs or PCA can make the network training easier and generalization better, but on the other hand the rules extracted from such networks may be more difficult to understand and to draw conclusions.

The possibility of obtaining the rules enhances the value of a classifier, because the user is provided not only with the final decision but also with the explanation how the decision was reached. The value of the classifier can be still more enhanced, when additional information about the probability of the decision being correct could be provided. In MLP networks, the probability of a given vector being assigned to each class can be considered proportional to this class output neuron signal (chapter 1.5-1.7).

In SMLP networks the output signals are either 0 or 1 and the probability cannot be obtained directly. However, some additional information can be provided, such as the crossvalidation accuracy for vectors classified by the same rule as the actual test vector, or the distance from the test vector to the class boundaries if the original features were continuous.

Theoretically it may happen that changing two weights at a time will not be sufficient for SMLP-DS convergence. Then the natural solution is to use SMLP-VSS, nevertheless some other training methods can also be considered. Changing three weights at time is very costly - $O(w^3)$, and frequently genetic algorithms (chapter 2.1.2.4) may be able to find the solution in fewer steps then changing three weights at a time through all the possible weight triples (excluding obviously the same feature weights from a simultaneous change). The network weights can be coded into chromosomes and a standard genetic optimization can be performed. However, genetic algorithms change all the weights at once, therefore the signal table cannot be used. Therefore, genetic algorithms will not necessarily be quicker than changing three weights at a time, in spite of fewer steps.

3.2.11. SMLP Architecture for Complex Rules

It is not always the best idea to use the M-of-N or the standard disjunctive normal form of rules, since sometimes it may lead to too complex and too long rules. Then a better solution maybe to add a single neuron in an additional network layer than many neurons in a single hidden layer. This problem may appear if the partial rules generated by particular hidden neurons must joined with the AND operator, for example:

(A or B) and (C or D)

The standard SMLP structure cannot perform this operation if (A, B) and (C, D) belong to different features, because it has not enough layers. The output layer performs OR operations, which cannot be changed to AND operations by changing the bias of the neuron, because this would not allow for adding any more partial rules joined with the OR operator. The solution is to add locally one neuron in the layer between the hidden and output layer. Theoretically, the data could be described by rules that could require *n* layers of neurons – then this approach can be extended to *n* layers.

The layers perform the following operations:
Layer[0] - provide feature values
Layer[1] - groups the values of the same feature together
Layer[2] - generates partial rules
Layer[3] - combines partial rules with AND operation
Layer[4] - combines partial rules with OR operation into classes

The standard SMLP-DS and SMLP-VSS procedures can be used to train such a network. It is a constructive solution and additional neurons are added only locally as needed.

Some experiments with a version of SMLP network, which changes the number of layers dynamically, adjusting its structure to the data, were performed. The network structure

is shown in a five-layer layout in Fig. 3.11. The zero layer is derived from the first one, like in Fig. 3.5-right, in order to make it clear how the operations are performed. The network has initially three layers (Layer 0, 1 and 4), what is sufficient for such data as Iris. As long as one hidden neuron is sufficient per given class, the hidden neuron performs the functions of both the hidden and output neuron in the standard SMLP network. If proper rules cannot be generated within the actual network structure, than Layer[3] with initially two neurons is added for the actual class (required for such data as Mushrooms). If proper rules still cannot be generated within the actual network structure, even with many hidden neurons, then Layer[2] with initially two neurons is added for the actual hidden (Layer[3]) neuron (For example, a dataset that requires this layer may describe a student taking an exam. The exam consists of 5 questions. In order to pass the exam, the student must answer question A and at least two of the remaining four questions. Another choice is to bribe the examiner. Thus, the rule will be: *if ((A=1) AND (2 of (B,C,D,E) = 1)) OR (bribe=1) then pass*)

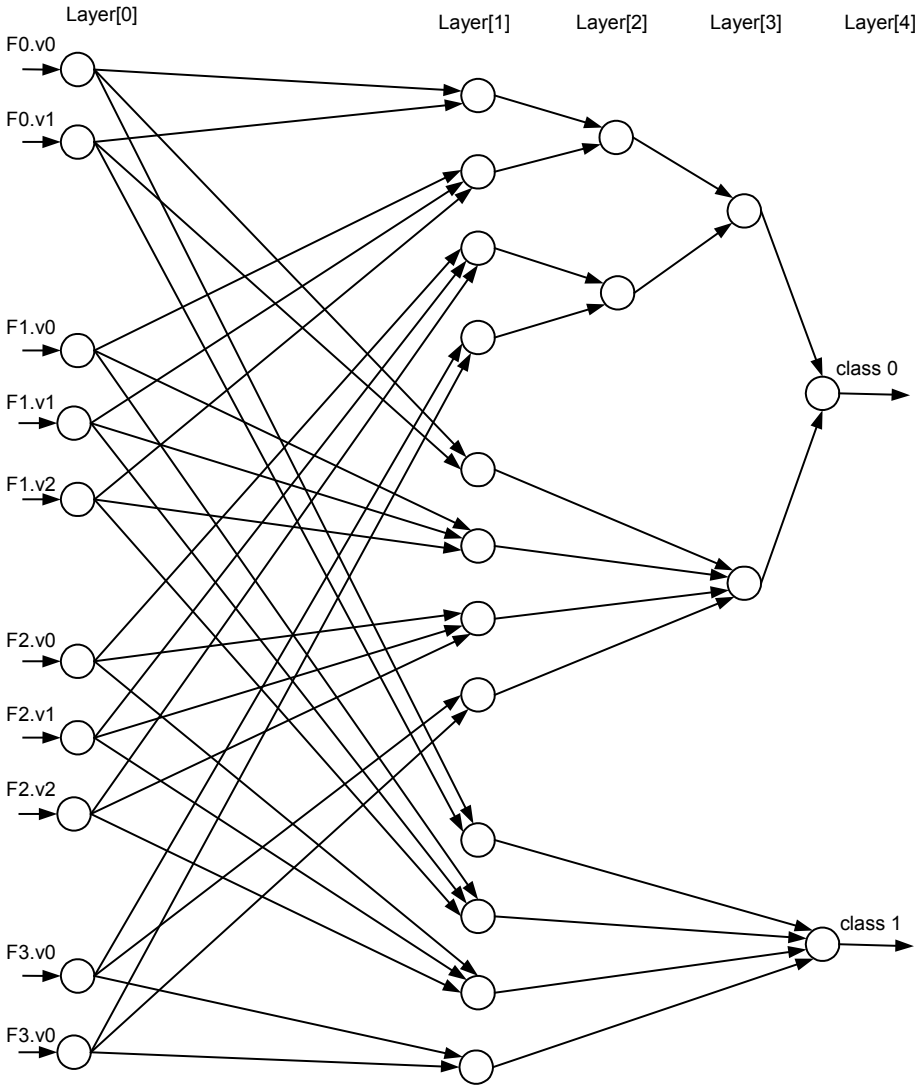

Fig. 3.11. A generalized SMLP network structure for complex data.

3.2.12. Experimental Results and Rules Extracted from Data

3.2.12.1. Criteria of Classifier Quality

The first criterion, which assesses the classifier quality, is the classification accuracy. Other parameters include the complexity of the algorithm, convergence properties (percentage of runs the algorithm converges), stability measured by a change of results when small perturbation of vectors located close to decision boundaries occurs, sensitivity, specificity, variance of results, comprehensibility of the rules, training times, memory requirements and additional information that the classifiers give besides the predicted class membership. All the above except for sensitivity and specificity were discussed in various chapters of this thesis.

A confusion matrix $\boldsymbol{C}$ is a square matrix that describes the errors made by a classifier. Each row i corresponds to a class the instances belong to and each column j to a class the instances were classified to. Thus, the element c_{ij} indicates the number of instances belonging to class C_i that were recognized as instances of class C_j. An example of a confusion matrix:

$i\backslash j$	C_1	C_2	C_3	C_4
C_1	100	3	2	3
C_2	2	80	0	4
C_3	0	2	60	2
C_4	0	6	2	70

Sensitivity describes the ability of a classifier F to detect a given class instances in the dataset X. Sensitivity $Se(C_i,F,X)$ is a conditional probability of an instance $x \in X$ being classified to class C_i by the classifier F, given that it really belongs to class C_i, and it can be obtained from the confusion matrix:

$$Se(C_i, F, X) = \frac{c_{ii}}{\sum_j c_{ij}} \tag{3.24}$$

Specificity describes the ability of a classifier F to reject the instances from other classes. Specificity $Sp(C_k,F,X)$ is a conditional probability of an instance $x \in X$ not being classified to class C_k by the classifier F, given that it really does not belong to class C_k and it can be obtained from the confusion matrix:

$$Sp(C_k, F, X) = \frac{\sum_{i \neq k} \sum_{j \neq k} c_{ij}}{\sum_{i \neq k} \sum_{j} c_{ij}} \tag{3.25}$$

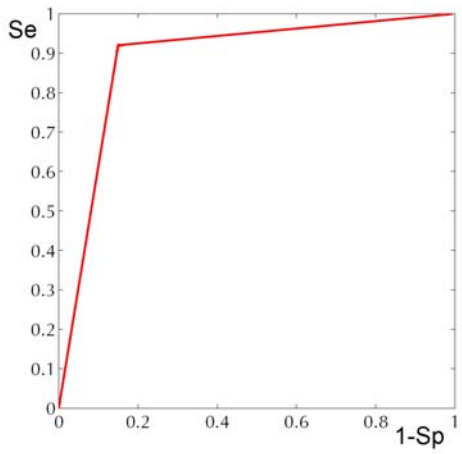

Fig. 3.12. ROC Curve.

The ROC curves (Receiver Operator Characteristic) can be used for the visual assessment of the classifier performance and especially for the comparison of several classifiers [Mertz 1978]. Each point on the ROC curve corresponds to the mean sensitivity and specificity of the classifier for all the classes. The sensitivity (Se) is on the vertical axis and 1-specificity (1-Sp) is on the horizontal axis. Bigger the areas under the ROC curve indicate higher quality of the classifier.

In cases, where there are different costs of misclassifying different class instances, the costs may be defined in the cost matrix $\dot{\boldsymbol{C}}$. For example the cost of classifying a poisonous mushroom as edible may be higher then the cost of misclassifying an edible mushroom as poisonous. An example of a cost matrix:

$i\backslash j$	$\dot{C}_1$	$\dot{C}_2$	$\dot{C}_3$	$\dot{C}_4$
$\dot{C}_1$	0	3	2	3
$\dot{C}_2$	2	0	5	4
$\dot{C}_3$	2	2	0	2
$\dot{C}_4$	2	6	2	0

where i is the original class and j is the predicted class. The task in the N-class problem is to minimize to total misclassification cost E expressed by sum of products of the appropriate entries in the confusion matrix and in the cost matrix:

$$E = \sum_{i=1}^{N} \sum_{j=1}^{N} \dot{c}_{ij} c_{ij} \tag{3.26}$$

A comparison of all the parameters for so many algorithms would be very difficult, especially that such parameters are only rarely available in the literature. Such a detailed comparison was even not contained in Statlog, a large-scale European project comparing classification algorithms [Statlog 1994], though much additional information can be found there.

For the datasets on which the algorithms were compared in the Statlog project, usually the longest training times were required by MLPs trained with backpropagation (MLP BP),

SOM, ALLOC80, SMART, AC^2 and cascade correlation networks. As it can be concluded from the presented results (and as it is known from the experience with the Ghostminer program [Ghostminer], where the incNet network is implemented), the training times of the incNet network, which performed well in some cases, are several ranks of orders longer than that of VSS, NG and SMLP. FSM performs best when it is used as a committee of networks, which makes the complexity of the model higher. PVM performs a complete search through the solution space and therefore for bigger dataset it is a very costly method.

In cases where the user must understand decisions of the classifier, the comprehensibility of rules can be even a more important factor than a very high accuracy on the test set (chapter 3.1.4).

NG and VSS are not self-standing classifiers but only training algorithms for MLP networks and must be evaluated together with the underlying architecture, because their performance is bound by the limitation of MLP network. NG and VSS should be rather compared with other MLP training algorithms, and they have been in chapter 2.4.5. It should be also pointed out that the MLP trainings were performed on raw data, where the only preprocessing was standardization of continuous features. Many other classifiers used more sophisticated data preparation techniques. For example, the classification was frequently preceded by feature selection and data discretization, which removed much noise from the data and reduced the search space.

3.2.12.2. Testing Procedure

Results obtained with NG, VSS and SMLP are compared to the best classification methods that can be found in the literature. The classification algorithms compared here were shortly introduced in chapter 3.1. Datasets selected for this comparison have been analyzed by many methods and crossvalidation or test accuracies obtained with the methods are available in the literature. Only the methods for which the results were available in the appropriate form are included in comparison. For example, several authors tested their methods on the original dataset divided into a separate training set and test set. In this case, the results strongly depend on the method (which was usually not reported) of dividing the dataset into the training and test set. Thus, because of different testing conditions the results cannot be compared with other methods, where crossvalidation was used and therefore they are not included in comparisons.

One should remember that the testing procedure is frequently not performed correctly. “The next point is that a real test set is a dataset that the classifier has never seen before. A frequent practice is however, to train the classifier on one set and then check its performance on another set (called by us “test set”). If the results are not satisfactory, then we change something in the algorithm and once again train it one the first set and test it on the second set. And so on. By such modifications we adjust the algorithm to the only test set we have. So the “test set” is really no longer a test set, but rather the second training set. Then we boast that our algorithm achieved 100% accuracy on the test set. In this context, it is rather advocated to use crossvalidation. Since with crossvalidation we have 10 different training and test sets, moreover they are different at each run of the algorithm, thus the algorithm is less prone to adjusting to a given test set.” (Norbert Jankowski at Bioinformatics Workshop, Toruń, 03 July 2004).

If the data preprocessing is performed on the entire set, the crossvalidation results will be overestimated. On the one hand, this would allow assessing the performance on the classifier alone, but on the other hand, the classifier will never be used alone in cases, where the data must be preprocessed. Thus, in crossvalidation tests, all the preprocessing of data, such as normalization, discretization or feature selection should be performed only on the training partition of the set. Then the validation partition should be transformed using the parameters determined on the training partition. This allows for testing the entire model and all the experiments presented in this thesis were conducted in this way.

It seems reasonable that for dataset with unequal class distribution rather the balanced accuracy should be maximized. (Also the accuracy given by the misclassification cost matrix can be maximized.) However, in the methods available for comparison, always the standard accuracy was maximized and reported. Therefore, to ensure the proper comparison with other results also the standard accuracy was maximized in my tests. The balanced accuracies presented in the tables were calculated using the equation (1.25).

The rules extracted from SMLP networks were always obtained on the entire training set. The accuracy of rules on the training and on the test set (if test set is available) is given. Moreover, the stability of rules and accuracy in crossvalidation tests are discussed. The accuracies on training dataset (or on the training partition of the dataset for crossvalidation) given in the tables for SMLP, NG and VSS are those accuracies that allowed for the highest test accuracies (chapter 2.6.1 – Fig. 2.48-right) and not the highest accuracies possible to obtain on the training set. It is almost always possible to obtain 100% accuracy on the training set (excluding the cases, where two identical vectors belong to different classes), but such networks generalize poor and the test accuracy falls down dramatically.

The 10-fold crossvalidation was run 10 times (together 100 trainings and 100 tests). If the test set was used, the training and test was performed only once for SMLP, because SMLP is a deterministic method and 10 times for NG and VSS starting from different random weights.

The standard deviation of the test accuracy within a single crossvalidation was calculated as

$$stdS = \sqrt{\frac{1}{100}\sum_{i=1}^{100}\left(acc_i - \overline{acc}\right)^2} \tag{3.27}$$

where the crossvalidation accuracy *acc* is the ratio of correctly classified vectors in the validation part of the dataset to the number of vectors in the validation part and $\overline{acc}$ is the quotient of the total number of correctly classified vectors in the experiment to the 10-fold number of vectors in the dataset.

The standard deviation between the mean accuracies of the whole crossvalidations was calculated as

$$stdCv = \sqrt{\frac{1}{10}\sum_{i=1}^{10}\left(acv_i - \overline{acc}\right)^2} \tag{3.28}$$

where the crossvalidation accuracy *acv* is the ratio of correctly classified vectors in the single 10-fold crossvalidation to the number of vectors in the training set.

The standard deviation provides not only information about the classifier stability but perhaps even more information about the dataset properties. SMLP is a fully deterministic method and for the same training and test sets always gives the same results. Nevertheless, the standard deviations for SMLP are sometimes as high as 10%. For example, there are 100 vectors in the training set and the classifier classifies 75% of them correctly and always classifies the same vectors wrongly no matter what is the division between the training and test set. In the best case, the vectors can be distributed in this way, that in half of the crossvalidation sets 7 of them are classified correctly and in the other half 8. Then the within crossvalidation standard deviation *stdS* will be 5.27% and there is no way to decrease it. But this does not tell us anything about the classifier stability. The between crossvalidation standard deviation *stdCv* also will never be zero even for a deterministic classifier, because there are different vectors in particular training and test crossvalidation partitions, however this value can be used to assess how much a particular result can differ from the mean value and in this aim it is provided in the tables. The standard deviation can be used as an absolute classifier stability measure only if all the trainings and tests are carried out on the same two sets.

Symbols used in the tables:

TS – separate test set
10CV – 10-fold crossvalidation
5CV – 5-fold crossvalidation
12CV – 12-fold crossvalidation
L1O – leave one out
1ch – one weight was changed at a time
2ch – two weights were changed at a time
BS – method based on beam search at feature level
sdtS – standard deviation of each test accuracy within a single crossvalidation
StdCv – standard deviation between the mean accuracies of the whole crossvalidations
x-x-x – structure of the network (number of neurons in the successive layers)
tc – number of training cycles
CN – number of networks in a committee
r – value of a regularization term in the error function $E = \sum_v \sum_c (d_{v,c} - s_{v,c})^2 + r\sum_i w_i^2$

FG, FR , FT ,R – gaussian, rectangular, triangular and rotation of transfer functions

3.2.12.3. Appendicitis

The dataset was donated by prof. Shalom Weiss from Rutgers University. The purpose of the analysis is to predict whether the patient suffers from appendicitis. There are 106 vectors, 21 (19.8%) in the first (no-appendicitis) class and 85 (80.2%) in the second (appendicitis) class. The dataset contains 7 continuous features, values of medical tests:

f1: WBC1
f2: MNEP
f3: MNEA
f4: MBAP
f5: MBAA
f6: HNEP
f7: HNEA

Logical rules were obtained with 1 hidden neuron per class. With more hidden neurons per class more detailed rules are obtained, covering correctly more training vectors, however they already overfit the data (using the more complex rules leads to lower accuracy in crossvalidation tests), thus it is not advocated to use them. Although the best result quoted in Table 3.8 was found by the IncNet neural network with 30 neurons, this was obtained with a high crossvalidation variance and the accuracy was higher in the test than on training set thus the best stable solution in this case is that of PVM.

Table 3.8. Classification results for the Appendicitis dataset.

method	training %	test %	test method	source
incNet (1100 epochs, 30 neurons)	90.1	90.9	10CV	[Jankowski 2003]
PVM	91.5	89.6	L1O	[UMK-KMK]
SSV – beam search	94.3	88.7	L1O	[Grąbczewski 2003]
SSV	94.3	88.7	L1O	[Grąbczewski 2003]
6-NN	-	88.0	10CV	[UMK-KMK]
FSM (FG+R+CN=20)	-	87.6	10CV	[Adamczak 2001]
FSM (FG+R)	-	86.2	10CV	[Adamczak 2001]
MLP BP	-	83.9	10CV	[UMK-KMK]
CART	90.6	84.9	L1O	[UMK-KMK]
Naive Bayes	88.7	83.0	10CV	[UMK-KMK]
C-MLP2LN, 1 neuron	91.5	-	L1O	[UMK-KMK]
C-MLP2LN, 2 neurons	94.3	-	L1O	[UMK-KMK]
default		80.2		
NG (7-1, 10tc)	89.6	87.5	10CV	this work
VSS (7-1, 5tc, r=0.2)	89.8	88.0	10CV	this work
SMLP-DS (35-1-1, 1ch, 5ed)	92.2	88.2	10CV	this work
SMLP-VSS (35-1-1, 5ed)	90.8	87.3	10CV	this work

Table 3.9. Additional parameters of the Appendicitis dataset training.

method	%test	%stdCv	%stdS	%test balanced	mean values in confusion matrix
NG	87.5	0.6	11.1	77.3	12.7 8.3 5.0 80.0
VSS	88.0	0.7	8.7	76.8	12.2 8.8 3.9 81.1
SMLP DS	88.2	1.1	9.6	74.5	10.9 9.1 2.4 82.6
SMLP VSS	87.3	1.2	10.8	73.6	10.7 10.3 3.2 81.8

Before each crossvalidation run, the cut-off points were determined by dividing each feature into 5 equal width intervals. Thus, in particular crossvalidation runs they could slightly differ from the values presented in the rules. There was no further optimization of the cut-off points (The optimization was attempted but it did not improve the results). The weights were changed one at a time or two at a time with SMLP-DS or SMLP-VSS was used; all the methods produced similar accuracy. Depending which weight was changed as first, different rules were obtained:

Rule 1: *if hnea<5570 then no-appendicitis else appendicitis*
(accuracy: 88.7%, sensitivity: 61.9%, specificity: 95.5%)

Rule 2: *if mnea<6670 then no-appendicitis else appendicitis*
(accuracy: 87.7%, sensitivity: 71.4%, specificity: 91.8%)

Rule 3: *if wbc1<8500 then no-appendicitis else appendicitis*
(accuracy: 87.7%, sensitivity: 57.1%, specificity: 94.1%)

Rule 4: *if mnea<6670 and mbap<12.1 then no-appendicitis else appendicitis*
(accuracy: 91.58%, sensitivity: 61.9%, specificity: 98.8%)

Rule 5: *if (wbc1<8500 or mnep<66) and mbap<12.1 then no-appendicitis else appendicitis*
(accuracy: 92.5%, sensitivity: 66.7%, specificity: 98.8%)

The, rules represent alternative ways to understand the structure of the data and depending on the costs of medical tests experts may prefer one rule to the others. In crossvalidation tests, one of the above rules or another rule combining two features was generated.

Three different rules (1,2,3) cover almost the same number of instances. Combining these rules, additional information can be obtained:

Rule 6: *if (hnea<5570 and mnea<6670) then no-appendicitis* (accuracy 89.2%)
if (hnea>5570 and mnea>6670) then appendicitis
if (hnea<5570 xor mnea<6670) then P(no-appendicitis) = P(appendicitis)=0.5

Although the total accuracy of rule 6 is slightly lower than that of rule 4 and 5, it probably better describes the properties of this dataset, providing more information about the structure of the data, as can be seen in Fig. 3.13. A forest of SMLP networks trained with different order of weight examination and feature selection methods can be created to provide sets of equivalent rules, so that more information about the data can be obtained or the form of rules that experts find more interesting can be chosen. Fuzzy rules can describe points in areas where crisp rules overlap. The value of the membership function of such a point can be proportional either to the probability density for a given class in this area or to the distance from that point to the decision border.

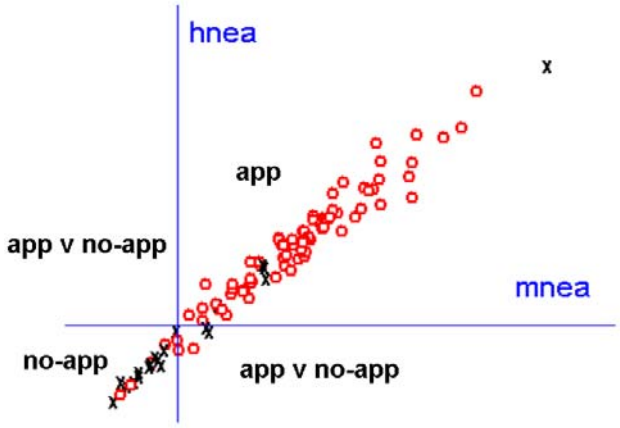

Fig. 3.13. The Appendicitis dataset with decision borders. Projection into two-feature space.

3.2.12.4. Wisconsin Breast Cancer

This dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg and is publicly available at UCI [Mertz 1998]. The purpose of the analysis is to predict whether the patient suffers from a benign or malignant breast cancer. There are 699 vectors, 458 (65.5%) in the first class (*benign*) and 241 (34.5%) in the second class (*malignant*). The first feature is a record label, the remaining 9 features have been discretized into 10 bins:

f1: Sample code number
f2: Clump Thickness
f3: Uniformity of Cell Size
f4: Uniformity of Cell Shape
f5: Marginal Adhesion
f6: Single Epithelial Cell Size
f7: Bare Nuclei
f8: Bland Chromatin
f9: Normal Nucleoli
f10: Mitoses

Table 3.10. Classification results for the Wisconsin Breast Cancer dataset.

method	training %	test %	test method	source
IncNet (3000 epochs, 40 neurons)	97.6	97.1	10CV	[Jankowski 1999]
3-NN, Manhattan	-	97.1	10CV	[UMK-KMK]
20-NN, Euclides	-	96.9	10CV	[UMK-KMK]
FDA	-	96.7	10CV	[Ster 1996]
MLP BP	-	96.7	10CV	[Ster 1996]
FSM (FG+R+CN=30)	-	96.6	10CV	[Adamczak 2001]
LVQ	-	96.6	10CV	[Ster 1996]
Naive Bayes	-	96.4	10CV	[UMK-KMK]
SSV	-	96.3	10CV	[Grąbczewski 2003]
LDA	-	96.0	10CV	[Ster 1996]
QUEST	-	95.9	10CV	[Lim 2000]
FSM (FR)	-	95.4	10CV	[Adamczak 2001]
C4.5	-	94.7	10CV	[Zarndt 1995]
CART	-	93.5	10CV	[Zarndt 1995]
default		65.5		
NG (10-2-1, 6tc, r=0.5)	97.2	96.9	10CV	this work
VSS (10-2-1, 4tc, r=0.5)	97.2	96.8	10CV	this work
SMLP-DS (97-1-1, 1ch-BS)	97.9	97.1	10CV	this work
SMLP-VSS	97.8	97.1	10CV	this work

Table 3.11. Additional parameters of the Wisconsin Breast Cancer dataset training.

method	%test	%stdCv	%stdS	%test balanced	mean values in confusion matrix
NG	96.9	0.69	1.9	96.6	446.7 11.3 10.5 230.5
VSS	96.8	0.20	1.7	96.6	445.3 12.7 9.7 231.3
SMLP DS	97.1	0.23	2.0	96.6	449.9 8.1 12.3 228.7
SMLP VSS	97.1	0.58	2.0	97.0	445.4 12.6 8.1 232.9

All rules were found using one hidden neuron and changing one weight at a time with SMLP-DS. The single zero weights surrounded by two +1 or two –1 weights of the same feature were automatically replaced by +1 or –1 weights respectively to remove the discontinuity. The third rule was found using beam search at feature level with changing one weight at a time.

if f3 <3.5 then benign else malignant
(accuracy: 92.7%, sensitivity: 96.9%, specificity: 84.7%)

if f2<6.5 and f3<3.5 then benign else malignant
(accuracy: 95.1%, sensitivity: 96.7%, specificity: 92.1%)

if f2<6.5 and f7<3.5 and f9<2.5 then benign else malignant
(accuracy: 98.0%, sensitivity: 98.9%, specificity: 96.3%) This rule is very stable, it is generated in almost every crossvalidation run giving on average 97.9% accuracy on the training partition and 97.1% on the test partition of the dataset.

It is worthwhile to note that the rules found here are both simpler and more accurate than those found by CART, C4.5 and SSV decision trees.

In the coordinate system of sum *S* and normalized product *NP* of all the features except the first one (which is the sample code number), two clusters of data corresponding to classes are clearly visible (Fig. 3.14). The visible separation in the space between the two classes leaves 20 vectors on the wrong side, which gives 97.14% accuracy. The best classifiers are asymptotically approaching this level.

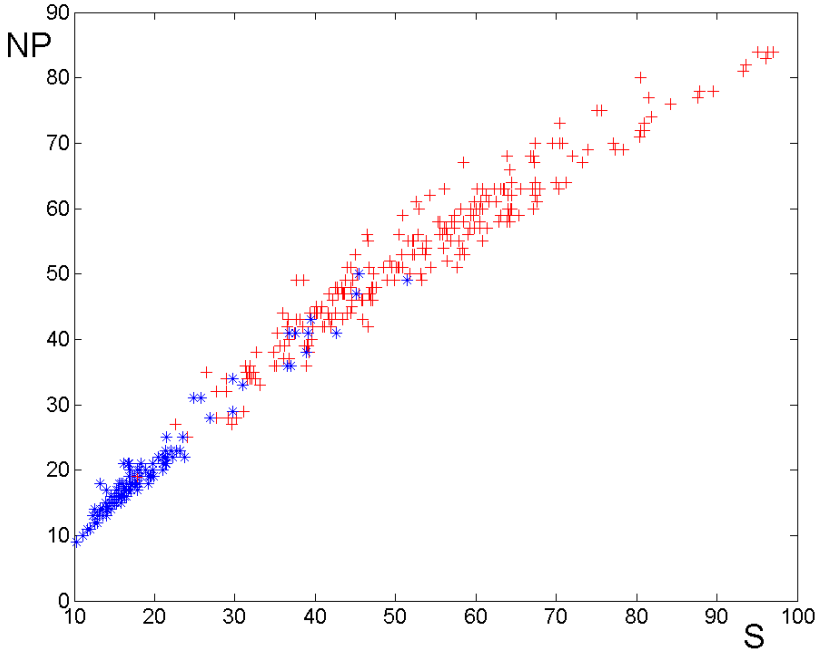

Fig. 3.14. Projection of the Wisconsin Breast Cancer dataset into the normalized product *NP* and sum *S* coordinate system. ($NP=const\cdot(f2\cdot\ldots\cdot f10)^{1/9}$, *1/9* is used in the power exponent, because there are 9 features, *const* is a normalization factor, to make the mean value of *NP* equal to the mean value of *S*.)

3.2.12.5. Thyroid

The dataset is publicly available at UCI [Mertz 1998]. The purpose of the analysis is to predict whether the patient suffers from primary hypothyroid, compensated hypothyroid or is healthy (no hypothyroid) given the results of various medical tests carried out on the patient. The training set contains 3772 vectors: 93 (2.5%) in the first class (*primary hypothyroid*) and 191 (5.1%) in the second class (*compensated hypothyroid*) and 3488 (92.4%) in the third class (*no hypothyroid*). The test set contains 3428 vectors: 73 (2.1%) in

the first class and 177 (5.2%) in the second class and 3178 (92.7%) vectors in the third class. The dataset contains 21 features, 6 continuous and 15 binary:

f1: age (continuous)
f2: sex (binary)
f3: treatment with thyroxine (binary)
f4: previous treatment with thyroxine (binary)
f5: treatment with antithyroid (binary)
f6: sick (binary)
f7: pregnant (binary)
f8: thyroid-surgery (binary)
f9: treatment with iodine 131- (binary)
f10: test for hypothyroid (binary)
f11: test for hyperthyroid (binary)
f12: treatment with lithium (binary)
f13: goiltre (binary)
f14: tumor (binary)
f15: hypopituitary (binary)
f16: psychological symptoms (binary)
f17: TSH level (continuous)
f18: T3 level (continuous)
f19: TT4 level (continuous)
f20: T4U level (continuous)
f21: FTI level (continuous)

Table 3.12. Classification results for the Thyroid dataset.

method	training	test	test method	source
PVM	99.79	99.33	TS	[Weiss 1990]
SSV	99.79	99.33	TS	[Grąbczewski 2003]
incNet (200 000 epochs, 9 neurons)	99.68	99.24	TS	[Jankowski 1999]
C4.5	-	99.2	TS	[Zarndt 1995]
FSM (FR+CN=20)	-	99.1	TS	[Adamczak 2001]
QUEST	-	99.1	TS	[Lim 2000]
CART	-	99.1	TS	[Zarndt 1995]
C-MLP2LN	99.86	99.07	TS	[UMK-KMK]
FSM (FR)	-	99.0	TS	[Adamczak 2001]
ID3	-	98.7	TS	[Zarndt 1995]
cascade correlation	100	98.48	TS	[Schiffmann 1993]
MLP + BP + genetic opt.	99.4	98.4	TS	[Schiffmann 1993]
1-NN, Euclides	98.4	97.7	TS	[UMK-KMK]
3-NN, Euclides	98.7	97.9	TS	[UMK-KMK]
MLP + BP	99.1	97.6	TS	[Schiffmann 1993]
Naive Bayes	97.03	96.06	TS	[UMK-KMK]
LDA	-	93.81	TS	[Lim 2000]
CAL5	-	92.74	TS	[Lim 2000]
default		92.40		
VSS (21-6-3, 40tc)	99.68	98.95	TS	this work
SMLP-DS 2x(x-1-1, 1ch)	99.79	99.33	TS	this work

Table 3.13. Additional parameters of the Thyroid dataset training.

method	%test	%std test	%test balanced	mean values in confusion matrix
VSS test	98.95	0.12	97.84	68 1 4 0 175 2 8 21 3149
SMLP DS training	99.79	0.00	99.23	91 0 2 0 191 0 3 3 3482
SMLP DS test	99.33	0.00	98.88	71 0 2 0 177 0 10 11 3157

First the continuous features were discretized. Then instead of a simple feature ranking, a feature filter based on SSV criterion [Grąbczewski 2003] was used. The filter determined the following feature order according to decreasing mutual information: TSH, th-surgery, FTI, on-thyroxine, TT4, pregnant, I131_treatment, query-hyperthyroid, lithium, tumor and other features on further positions. The network training was performed on the two first classes separately using initially the balanced error and adjusting the cut-off points after each training cycles. Afterwards the error was changed to the standard error (that mainly shifted the cut-off point value for the TSH feature). The following rules were found changing one weight at a time with one hidden neuron per class:

if TSH>0.0061 and FTI<0.0647 and th-surgery=no then primary hypothyroid
(training: sensitivity: 97.85%, specificity: 99.92%, test: sensitivity: 97.26%, specificity: 99.70 %)
if TSH>0.0061 and FTI>0.0647 and TT4<0.15 and on-thyroxine=no and th-surgery=no then compensated hypothyroid
(training: sensitivity: 100%, specificity: 99.92%, test: sensitivity: 100%, specificity: 99.66%)
else no hypothyroid
(training: sensitivity: 99.83%, specificity: 99.30%, test: sensitivity: 99.34%, specificity: 99.20%)

(training accuracy: 99.79%, test accuracy: 99.33%)

The decision borders are almost ideally hyperrectangular and therefore the SMLP network can obtain very high accuracy with only one hidden neuron per class. The standard MLP network requires much more hidden neurons with sigmoidal transfer functions to approximate the decision borders with comparable accuracy.

3.2.12.6. Ljubljana Breast Cancer

This breast cancer database from the University Medical Center, Institute of Oncology, Ljubljana, Yugoslavia was donated by M. Zwitter and M. Soklic and is publicly available at UCI [Mertz 1998]. The purpose of the analysis is to predict whether the patient suffers from recurrent or no-recurrent breast cancer. There are 286 vectors, 201 (70.2%) in the first (no-recurrent) class and 85 (29.8%) in the second (recurrent) class. The dataset contains 9 discrete features, some of them were originally continuous but are available only in the discretized form:

f1: age
f2: menopause
f3: tumor-size
f4: involved-nodes
f5: node-caps
f6: degree-malignant
f7: breast
f8: breast-quad
f9: irradiation

Table 3.14. Classification results for the Ljubljana Breast Cancer dataset.

method	training	test	test method	source
C-MLP2LN	78.0	77.4	10CV	[UMK-KMK]
PVM	77.4	77.1	10CV	[Weiss 1990]
MML	-	75.3	10CV	[Zarndt 1995]
C4.5	-	73.9	10CV	[Zarndt 1995]
MLP BP	-	73.5	10CV	[Zarndt 1995]
SSV	-	72.7	10CV	[Grąbczewski 2003]
AQ15	-	72.0	10CV	[Statlog 1994]
FSM (FG+R)	-	71.6	10CV	[Adamczak 2001]
CART	-	71.4	10CV	[Zarndt 1995]
CN2	-	70.7	10CV	[Zarndt 1995]
Naive Bayes	-	69.3	10CV	[Zarndt 1995]
ID3	-	66.2	10CV	[Zarndt 1995]
default		70.2		
NG (51-2-1, 6tc, r=0.35)	78.0	75.9	10CV	this work
VSS (51-2-1, 3tc, r=0.5)	78.8	76.0	10CV	this work
SMLP-DS (51-1, 2ch)	78.0	76.0	10CV	this work
SMLP-VSS (51-1)	78.9	75.7	10CV	this work

The rules for the Ljubljana Breast Cancer dataset were obtained with one hidden neuron per class, changing one weight at a time with SMLP-DS and with SMLP-VSS.

if degree-malignant<3 than no-recurrent else recurrent
(accuracy: 72.0%, sensitivity: 80.1%, specificity: 52.9%)

if node-caps=no and degree-malignant=2 than no-recurrent else recurrent
(accuracy: 75.5%, sensitivity: 94.5%, specificity: 30.6%)

if involved-nodes>2 and degree-malignant>2 than recurrent else no-recurrent
(accuracy: 76.2%, sensitivity: 31.8%, specificity: 95.0%)

Table 3.15. Additional parameters of the Ljubljana Breast Cancer dataset training.

method	%test	%stdCv	%stdS	%test balanced	mean values in confusion matrix
NG	75.9	0.28	7.2	61.1	20.8 64.2 4.6 196.4
VSS	76.0	0.25	6.3	60.2	18.0 67.0 1.7 199.3
SMLP DS	76.0	0.45	8.0	61.4	21.8 63.2 5.6 195.4
SMLP VSS	75.7	0.92	8.3	60.5	19.6 65.4 4.0 197.0

3.2.12.7. Cleveland Heart Disease

This dataset comes from the Cleveland Clinic Foundation and is publicly available form the machine learning database repository at UCI [Mertz 1998]. The purpose of the analysis is to predict the presence or absence of the heart disease given the results of various medical tests carried out on a patient. There are 303 vectors, 165 (54.5%) in the first class (healthy) and 138 (45.5%) in the second class (sick). The dataset contains 13 features:

f1: age (continuous)
f2: sex (binary)
f3: CP- chest pain type (4 discrete values)
f4: restbps - resting blood pressure (continuous)
f5: chol - serum cholesterol in mg/dl (continuous)
f6: fbs - fasting blood sugar > 120 mg/dl (binary)
f7: restecg - resting electrocardiographic results (3 discrete values)
f8: thalach - maximum heart rate achieved (continuous)
f9: exang - exercise induced angina (binary)
f10: oldpeak - ST depression induced by exercise relative to rest (continuous)
f11: slope of the peak exercise ST segment (discrete)
f12: ca - number of major vessels colored by fluoroscopy (3 discrete values)
f13: thal (discrete)

Table 3.16. Classification results for the Cleveland Heart Disease dataset.

method	training %	test %	test method	source
LDA	-	84.5	10CV	[Ster 1996]
FDA	-	84.2	10CV	[Ster 1996]
Naive Bayes	-	83.4	10CV	[UMK-KMK]
FSM (FT+CN=20)	-	83.2	10CV	[Adamczak 2001]
LVQ	-	82.9	10CV	[Ster 1996]
FSM (FG+R)	-	82.5	10CV	[Adamczak 2001]
SVM	-	81.5	5CV	[Bennet 1997]
kNN	-	81.5	10CV	[Ster 1996]
MLP BP	-	81.3	10CV	[Ster 1996]
CART	-	80.8	10CV	[Ster 1996]
SSV	-	79.7	10CV	[Grąbczewski 2003]
RBF	-	79.1	10CV	[UMK-KMK]
ASR	-	78.4	10CV	[Ster 1996]
C4.5	-	77.8	5CV	[Bennet 1997]
QDA	-	75.4	10CV	[Ster 1996]
LFC	-	75.1	10CV	[Ster 1996]
ASI	-	74.4	10CV	[Ster 1996]
OC1	-	71.7	5CV	[Bennet 1997]
1R	-	71.0	10CV	[UMK-KMK]
FOIL	-	66.4	10CV	[UMK-KMK]
default		54.13		
NG (24-2-1, 8tc, r=0.5)	86.9	85.0	10CV	this work
VSS (24-2-1, 3tc, r=0.7)	87.7	86.1	10CV	this work
SMLP-DS (28-1-1, 1ch)	84.5	81.5	10CV	this work
SMLP-VSS (28-1-1)	87.2	85.5	10CV	this work

Table 3.17. Additional parameters of the Cleveland Heart Disease dataset training.

method	%test	%stdCv	%stdS	%test balanced	mean values in confusion matrix
NG	85.0	0.44	5.5	84.6	145.8 19.2 26.4 111.6
VSS	86.1	0.30	5.1	85.7	149.2 15.8 26.4 111.6
SMLP DS	81.5	1.6	7.4	81.1	142.2 22.8 33.2 104.8
SMLP VSS	85.5	0.57	6.0	85.0	149.8 15.2 28.8 109.2

Seven of the vectors originally contained one missing feature value, which was replaced by the average value for their class.

First the SMLP network was trained on each feature separately to build feature ranking. Various discretization methods gave various information gain of the continuous features, however the differences were not enough big to change any feature position in the ranking. The first seven features were: thal (76.90%), cp (75.91%), ca (74.92%), exang (71.95%), oldpeak (70.30%), thalach (69.64%), slope (69.31%). Then the features were reordered according to the ranking. One hidden neuron was used in all trainings. The following rules were obtained with SMLP-DS changing one weight at time as well as with SMLP-VSS:

if thal<>2 then healthy else sick
(accuracy: 76.9%, sensitivity: 79.4%, specificity: 73.9%) This rule is very stable, it is generated in each crossvalidation run giving on average 75.3% accuracy on the test part of the dataset.

if thal<>2 and cp<>2 and ca=0 then healthy else sick
(accuracy: 85.5%, sensitivity: 89.7%, specificity: 80.4%) This rule is relatively stable, it is generated (sometimes slightly modified) in most of crossvalidation runs.

3.2.12.8. Pima Indians Diabetes

This dataset was constructed by a selection from a larger database held by the National Institute of Diabetes and Digestive and Kidney Diseases. It is publicly available from the machine learning database repository at UCI [Mertz 1998]. The patients represented in this dataset are females at least 21 years old of Pima Indian heritage living near Phoenix. The problem posed here is to predict whether a patient would test positive for diabetes given a number of physiological measurements and medical test results. This as a two-class problem with class value 1 being interpreted as "tested positive for diabetes". The dataset has 768 vectors, 500 (65.1%) in the first class (no diabetes) and 268 (34.9%) in the second class (diabetes). The dataset is rather difficult to classify. The class value is really a binarised form of another attribute, which is itself highly indicative of certain types of diabetes but does not have a one-to one correspondence with the medical condition of being diabetic. The feature f1 is discrete, the other 7 features are continuous:

f1: number of times pregnant
f2: plasma glucose concentration in an oral glucose tolerance test
f3: diastolic blood pressure (mm/Hg)
f4: triceps skin fold thickness (mm)
f5: 2-hour serum insulin (mu U/ml)
f6: body mass index (kg/m^2)
f7: diabetes pedigree function
f8: age (years)

Table 3.18. Classification results for the Diabetes dataset.

method	training	test	test method	source
logDA	-	77.7	12CV	[Statlog 1994]
DIPOL92	-	77.6	12CV	[Statlog 1994]
incNet (5000 epochs, 100 neurons)	77.2	77.6	10CV	[Jankowski 1999]
LDA	-	77.5	12CV	[Statlog 1994]
SMART	-	76.8	12CV	[Statlog 1994]
QUEST	-	76.7	12CV	[Lim 2000]
RBF	-	75.7	12CV	[Statlog 1994]
FSM (FT+CN=20)	-	75.6	10CV	[Adamczak 2001]
ITRULE	-	75.5	12CV	[Statlog 1994]
MML	-	75.5	10CV	[Zarndt 1995]
FSM (FT)	-	75.2	10CV	[Adamczak 2001]
MLP BP	-	75.2	12CV	[Statlog 1994]
CAL5	-	75.0	12CV	[Statlog 1994]
SSV	-	74.8	12CV	[Grąbczewski 2003]
CART	-	74.7	10CV	[Zarndt 1995]
CASTLE	-	74.2	12CV	[Statlog 1994]
Naive Bayes	-	73.8	12CV	[Statlog 1994]
QDA	-	73.8	12CV	[Statlog 1994]
C4.5	-	73.0	12CV	[Zarndt 1995]
LVQ	-	72.8	12CV	[Statlog 1994]
SOM	-	72.7	12CV	[Statlog 1994]
AC^2	-	72.4	12CV	[Statlog 1994]
NewID	-	71.1	12CV	[Statlog 1994]
CN2	-	71.1	12CV	[Statlog 1994]
ALLOC80	-	69.9	12CV	[Statlog 1994]
kNN	-	67.6	12CV	[Statlog 1994]
default		65.1		
NG (8-2-1, 5tc, r=0.5)	77.8	77.0	10CV	this work
VSS (8-2-1, 3tc, r=0.5)	78.2	77.3	10CV	this work
SMLP-VSS (40-1, 3tc, r=0.5)	78.6	76.8	10CV	this work

Table 3.19. Additional parameters of the Diabetes dataset training.

method	%test	%stdCv	%stdS	%test balanced	mean values in confusion matrix
NG	77.0	0.40	4.2	72.6	155.6 112.4 64.0 436.0
VSS	77.3	0.26	4.8	72.9	156.0 112.0 62.2 437.8
SMLP VSS	76.8	0.30	4.1	72.0	151.0 117.0 61.6 438.4

The first four features in the feature ranking are: f2 (75.0%), f1 (67.8%), f8 (66.9%), f6 (66.3%). The following rules were obtained with one hidden neuron changing one weight at a time with SMLP-DS:

if f2>157 then no-diabetes else diabetes
(accuracy: 75.0%)

if f1>6.85 and f2<157 and f6>42 and 49<f8<70 then no-diabetes else diabetes
(accuracy: 80.6%)

3.2.13. Conclusions

A neural network approach to classification and rule extraction, called SMLP has been proposed. The model combines the advantages of MLP neural networks with the possibility of extracting simple rules in a comprehensive way. The training algorithms are much simpler than the gradient-based algorithms. Due to the perceptron properties, the rules given by hidden neurons are in the M-of-N form. Since the prepositional form of logical rules is usually preferred, M-of-N rules are reduced to AND + OR operations whenever possible.

As the experiments showed, the accuracy of results on the popular benchmark data sets is comparable with the best results obtained from other methods, while the algorithms are simple and computationally efficient. It cannot be said that the only criterion of the rule quality is the classification accuracy either using crossvalidation or a separate test set. Sometimes rules which are simpler or which better reflect the data structure may be preferred, although their accuracy is lower. It is possible to obtain several sets of rules by the modification of network parameters and training process. A forest of SMLP networks can be built to give users the possibility of choosing sets of rules that are most suitable for their purpose.

It seems that the search-based approach to logical rule extractions has a large potential worth further investigation.

4. Summary

Several properties of MLP networks were examined, including the properties of MLP error surfaces, learning trajectories, trends of weight changes and neuron signals. The PCA-based visualizations of many MLP error surfaces were presented and the factors influencing their properties were discussed. The possibility of training network in the reduced search space was discussed. The properties of error surface sections in different layer weight directions and in different phases of training were examined.

Basing on the conclusions from this research, two new MLP learning algorithms were developed: numerical gradient (NG) and variable step search algorithm (VSS). The algorithms do not impose any restrictions on network structure and neural transfer functions, which in particular do not have to be differentiable. The algorithms were tested on many datasets and a comparison including many factors with other MLP learning algorithm was presented on several datasets. Especially VSS algorithm proved to have the ability of finding good solutions, with very low network error at a low computational effort and with high stability of the achieved results. Several methods of reducing the computational costs and improving network generalization were discussed.

A search-based approach to logical rule extraction from MLP network with quantized parameters was presented. The logical rules are extracted from data by the analysis of the SMLP network weights. Two search-based algorithms from SMLP networks were proposed: the direct search method and a modified version of variable step search algorithms. Several additional aspects of these algorithms were discussed and possible solutions were proposed.

It should finally be concluded that the search-based algorithms can be successfully applied for multilayer perceptron training and for logical rule extraction from data using MLP networks. The proposed solutions in many aspects performed better than gradient-based optimization algorithms.

5. Future Work

- Search for other interesting projections of MLP error surface that will reveal more error surface properties, maybe some kinds of kernel PCA or other non-linear projections.
- Apply VSS also to MLP trained for regression problems.
- Find a more effective sequence of examining the weight changes in VSS.
- Analyze MLP decision borders and the ways to influence them more precisely than only by minimizing the global error measure.
- I have no plans concerning rule extraction systems, since the above-mentioned topics seem more interesting to me and moreover very many people work nowadays on rule extraction systems. So I leave this topic for them and wish them good luck.

6. List of Publications

- M. Kordos, W. Duch, "Search-based Training for Logical Rule Extraction by Multilayer Perceptron", Proc. of the Joint Int. Conf. on Artificial Neural Networks (ICANN) and Int. Conf. on Neural Information Processing (ICONIP), Istanbul, June 2003, pp. 86-89

- M. Kordos, W. Duch, "Multilayer Perceptron Trained with Numerical Gradient", Proc. of the Joint Int. Conf. on Artificial Neural Networks (ICANN) and Int. Conf. on Neural Information Processing (ICONIP), Istanbul, June 2003, pp. 106-109

- M. Kordos, W. Duch, "On Some Factors Influencing MLP Error Surface", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004, pp. 217-222

- M. Kordos, W. Duch, "Variable Step Search Algorithm for MLP Training", Proc. of the 8th IASTED Int. Conf. on Artificial Intelligence and Soft Computing, Marbella, Spain, September 2004, pp. 215-220

- M. Kordos, W. Duch, "A Survey of Factors Influencing MLP Error Surface", Control and Cybernetics, vol. 33, no. 4, 2004

- M. Kordos, "Directions in Multilayer Perceptron Weight Space", 4th Warsaw International Seminar on Soft Computing, Warsaw, October 25, 2004

- M. Kordos, "Search-based Approach to Multilayer Perceptron Training", Studia Informatica, vol. 26, no. 1 (62), 2005

7. References

[Adamczak 2000] R. Adamczak, W. Duch, "Neural Networks for Structure-activity Relationship Problmes"

[Adamczak 2001] R. Adamczak, "Zastosowanie Sieci Neuronowych do Klasyfikacji Danych Doświadczalnych", PhD Thesis, Uniwersytet Mikołaja Kopernika, Toruń

[Acid 1991] S. Acid et. al., "CASTLE: Casual structures from inductive learning. Release 2.0", Report No. 91-4-3, University of Granada

[Ajith 2002] Ajith Abraham, "Cerebral Quotient of Neuro Fuzzy Techniques – Hype or Hallelujah?", http://www.bytesforall.org/8th/abraham_bytes.pdf

[Altman 1994] E. I. Altman, G. Marco, F. Varetto, "Corporate Distress Diagnosis: Comparisons using linear discriminant analysis and neural networks (the Italian experience)", J. Bank. Finance vol. 18, pp. 505-529

[Andrews 1994] R. Andrews, S. Geva, "Rule Extraction from a Constrained Error Backpropagation MLP", 5th Aust. Conf. Neural Networks, pp. 9-12

[Andrews 1995] R. Andrews, J. Diederich, A. Tickle, "Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks", Knowledge-Based Systems, 8(6): 373-384

[Atkosoft 1997] Atkosoft S. A., "Survey on Visualization Methods and Software Tools", SUP.COM 96/Lot 30 Part A

[Azimi 1992] R. Azimi-Dadjahi, R. J. Liou, "Fast Learning Process of Multilayer Neural Networks Using Recursive Least Square Method", IEEE Transactions on Signal Processing, vol. 40, no. 2

[Battiti 1995] R. Battiti, G. Tecchiolli, "Training Neural Nets with the Reactive Tabu Search", Transactions on Neural Networks, vol. 6, pp. 1185-1200

[Bennet 1997] K. P. Bennet, J. A. Blue, "A Support Vector Approach to Decision Trees", RPI MAth Report, Rensselaer Polytechnic Institute, Troy, NY

[Berthold 1995] M. R. Berthold, K. P. Huber, "Extraction of Soft Rules from RecBF Networks" International Symposium on Intelligent Data Analysis (IDA-95), Baden-Baden, Germany, August 1995

[Bielecki 2004] A. Bielecki, P. Hajto, "A Neural-Based Agent for IP Traffic Scanning and Worm Detection", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004, pp. 816-822

[Bilski 2002] J. Bilski, L. Rutkowski, "Numerically Robust Learning Algorithms for Feed Forward Neural Networks", 6th Int. Conf. on Neural Networks and Soft Computing, Zakopane, Poland, June 2002, pp. 149-154

[Bilski 2004] J. Bilski, "Momentum Modification of the RLS algorithms", 7th Int. Conf. on Neural Networks and Soft Computing, Zakopane, Poland, June 2004, pp. 151-157

[Blum 1989] E. K. Blum, "Approximation of Boolean Functions by Sigmoidal Networks: Part I: XOR and other two-variable functions", Neural Computation, vol. 1, pp. 532-540

[Brenda 2002] Brenda Mak, Toshinori Munukata, "Rule Extraction from Expert Heuristics: A comparative study of rough sets with neural networks and ID3", European Journal of Operational Research, 136, pp. 212-229

[Breiman 1984] L. Breiman et. al. "Classification and Regression Trees", Wadsworth, Belmont, CA

[Brodley 1992] C. E. Brodley, P. E. Utgoff, "Multivariate versus univariate decision trees", Institute Raport 92-8, Department of Computer Science, University of Massachusetts

[Bullinaria 2002] John A. Bullinaria, "Introduction to Neural Networks", http://www.cs.bham.ac.uk/~jxb/inn.html
[Buntine 1993] W. Buntine, "Learning classification trees", Artifical Intelligence Frontiers in Statistics, Chapman & Hall, London, pp. 182-201
[Busse 1999] J. W. Grzymała-Busse, "LERS – a knowledge discovery system", Rough Sets in Knowledge Discovery, Physica-Verlag, Heidelberg, pp. 562-565
[Buzing 2001] Pieter Buzing, "Hybrid Systems: Two Examples of the Combination of Rule-Based Systems and Neural Nets", March 2001, http://www.cs.vu.nl/~pcbuzing/ Articles/hybrid.doc
[Chakraborty 2004] Debrup Chakraborty and Nikhil R. Pal, "A Neuro-Fuzzy Scheme for Simultaneous Feature Selection and Fuzzy Rule-Based Classification", IEEE Transactions on Neural Networks, vol. 15, no. 1, January 2004
[Cichosz 2000] P. Cichosz, "Systemy uczące się", WNT, Warszawa
[Clark 1987] P. Clark, T. Niblett, "Induction in Noisy Domains", Progress in Machine Learning, pp. 11-30, Bled, Yugoslavia, Sigma Press, 1987
[Clark 1989] P. Clark, T. Niblett, "The CN2 induction algorithm", Machine Learning 3(4), pp. 261-283
[Coetze 1997] F. M. Coetze, V. L. Stonick, "488 Solutions to the XOR Problem", Advances in Neural Information Processing Systems, vol. 9, pp. 410-416, Cambridge, MA, MIT Press, 1997
[Cottrell 1995] M. Cottrell et. al., "Neural Modeling for Time Series: a statistical stepwise method for weight elimination", IEEE Transactions on Neural Networks 6(6), pp. 1355-1364
[Craven 1996a] M. W. Craven, " Extracting Comprehensible Models from Trained Neural Networks", PhD Thesis, University of Wisconsin
[Craven 1996b] M. W. Cranven, J. W. Shavlik, "Extracting tree-structured representation of trained networks", Advances in Neural Information Processing Systems, vol. 8, pp. 24-30
[Decloedt 1996] L. Decloedt, F. Osorio, B. Amy, "RULE-OUT Method: A new approach for knowledge explication from trained ANN", Rule Extraction from Trained Artificial Neural NEtworks Workshop, pp. 34-42, Queensland University of Technology
[Delphi] http://www.borland.com/delphi/
[Denker 1987] J. Denker et. al., "Large automatic learning, rule extraction and generalization", Complex Systems, 1:887-922
[Dennis 1983] J. E. Dennis, R. B. Schnabel, "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Englewood Cliffs, NJ: Prentice-Hall
[Duch 1997] W. Duch, R. Adamczak, "New Developments in Feature Space Mapping Model", Third Conference on Neural Networks and their Applications, pp.65-70, Kule
[Duch 1997b] W. Duch, et. al., "Extraction of Crisp Logical Rules Using Constrained Backpropagation Networks – Comparison of Two Approaches", The European Symposium on Artificial Neural Networks, Bruge, pp. 109-114
[Duch 1999a] W. Duch, J. Korczak, "Optimization and global minimization methods suitable for neural networks", KMK UMK Technical Report 1/99
[Duch 1999b] W. Duch, N, Jankowski, "Survey of Neural Transfer Functions", Neural Computing Surveys 2, pp. 163-212
[Duch 1999c] W. Duch, K. Grąbczewski, "Searching for optimal MLP", The Fourth Conference on Neural Networks and Their Applications, Zakopane, May 1999, pp. 65-70

[Duch 2000] W. Duch, J. Korbicz, L. Rutkowski, R. Tadeusiewicz, "Sieci Neuronowe", Exit, Warszawa
[Duch 2001] W. Duch, R. Adamczak, K. Grąbczewski, "A New Methodology of Extraction, Optimization and Application of Crisp and Fuzzy Logical Rules", IEEE Transactions on Neural Networks 12, pp. 277-307
[Duch 2003b] W. Duch et. al., "Feature Selection and Ranking Filters", Proc. of Int. Conf. on Artificial Neural Networks (ICANN), Istanbul, June 2003, pp. 251-254
[Duch 2004a] W. Duch, "Visualization of Hidden Node Activity in Neural Networks", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004, pp. 38-49
[Duch 2004b] W. Duch, "Support Vector Neural Training", IEEE Transactions on Neural Networks, accepted November 2004
[Duch 2004c] W. Duch, R. Setiono, J. Żurada, "Computational Intelligence Methods for Rule-Based Data Understanding", Proceedings of the IEEE, May 2004
[Duch 2005] W. Duch, "Internal Representations of Multilayer Perceptrons", IEEE Transactions on Neural Networks, submitted January 2005
[Duch 2005b] W. Duch, "Uncertainty of data, fuzzy membership functions, and multi-layer perceptrons", IEEE Transactions on Neural Networks 16(1): 10-23
[Duda 2001] R. O. Duda, P. E. Hart, D. G. Stork, "Pattern Classification", New York, Wisley
[Engel 1988] J. Engel, "Teaching feed-forward neural networks by simmulaated annealing", complex systems 2, pp. 641-648
[Fang 1999] Fang Wang et al. "Neural Network Structures and Training Algorithms for RF and Microwave Applications"
[Fahlman 1991] S. E. Fahlman, C. Lebiere, "The Cascade correlation learning architecture", Neural Information Processing Systems, vol. 2, pp. 524-523
[Fahlman 1998] S. E. Fahlman, "Faster Learning Variations of Backpropagation: an empirical study", Connectionist Models Summer School, Morgan Kaufmann, pp. 38-51
[Finnhoff 1993] W. Finnhoff, F. Hergert, H. G. Zimmermann, "Improving Model Detection by Nonconvergent Methods", Neural Networks, 6(6), pp. 771-783, 1993
[Fisher 1936] R. A. Fisher, "The use of multiple measurements in taxonomic problems", reprinted in Contributions to Mathematical Statistics, John Wiley & Sons, New Yok, 1950
[Freund 1997] Y. Freund, R. E. Schapire, "A Decision Theoretic Generalization of On-line Learning and an Application to Boosting", Journal of Computer and System Sciences, 55(1):119-139
[Fu 1994], L. Fu, "Rule generation from neural networks", IEEE Transactions on Systems, Man and Cybernetics, vol. 28(8), pp. 1114-24
[Gabriel 2003] J. Gabriel, R. C. Gomes, Sanjit. K. Mitra, "Low-Complexity Image Compression Without A/D Conversion Using Analog Multilayer Perceptron", Proc. European Conference on Circuit Theory and Design (ECCTD), pp. III.281-III.284, Krakow, Poland, September 2003
[Gallagher 2000] M. Gallagher, "Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modeling", PhD Thesis, University of Queensland
[Gallagher 2003] M. Gallagher, T. Downs, "Visualization of Learning in Multi-layer Perceptron Networks using PCA", IEEE Transactions on Systems, Man and Cybernetics – part B: Cybernetics, vol. 33, pp. 215-221
[Garris 1998] M. S. Garris, C. L. Wilson, J. L. Blue, "Neural network-based systems for handprint OCR applications", Image Processing, vol. 3, no. 8

[Gaweda 2000] A. E. Gaweda, R. Setiono, J. M. Zurada, "Rule Extraction from Feedforward Neural Network for Function Approximation", 5th National Conference on Neural Networks And Soft Computing, Zakopane, Poland, June 6-10, 2000, pp. 311-316

[Ghostminer] GhostMiner data mining software, http://www.fqspl.com.pl/?a=product_view&id=2

[Goldberg 1989] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley

[Goodman 1989] R. M. Goodman, P. Smyth, "The induction of probabilistic rule sets – the ITRULE algorithm", The 6th International Workshop on Machine Learning, San Mateo, CA, pp. 129-132

[Gorban 1999] A. N. Gorban et. al., "Generation of Explicit Knowledge from Empirical Data through Pruning of Trainable Neural Networks", International Joint Conference on Neural Networks (IJCNN '99)

[Gori 1992] M. Gori, A. Tesi, "On the Problem of Local Minima in Backpropagation", IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-14, pp.76-86, 1992

[Grąbczewski 2003] K. Grąbczewski, "Zastosowanie kryterium separowalności do generowania reguł klasyfikacji na podstawie baz danych", PhD Thesis, IBS PAN, Warszawa, 2003

[Grinstein 2001] G. Grinstein, M. Trutschl, U. Cvek, "High-Dimensional Visualizations", 7th Data Mining Conference-KDD, 2001

[Halgamuge 1994] S. K. Halgamuge, B. J. McNeil, "Neural Networks in Designing Fuzzy Systems for Real World Applications", Fuzzy Sets and Systems, vol. 65, pp. 1-12, 1994

[Hamey 1995] L. G. C. Hamey, "The Structure of Neural Network Error Surfaces", The 6th Australian Conference on Neural Networks, University of Sydney, pp. 197-200, 1995

[Hamey 1998 89] L. G. C. Hamey, "XOR Has No Local Minima: A case study in neural network error surface analysis", Neural Networks, 11(4), pp. 669-681, 1998

[Hamilton 1996] H. J. Hamilton, N. Shan, N. Cercone, "RIAC: a rule induction algorithm based on approximate classification", Tech. Rep. CS 96-06, Regina University, 1996

[Hamilton 2002] H. J. Hamilton, "Knowledge Discovery in Databases", Department of Computer Science, University of Regina, http://www2.cs.uregina.ca/~hamilton/courses/831

[Hamm 2002] L. Hamm, B. Wade Brorsen, "Global Optimization Methods", The 2002 International Conference on Machine Learning and Applications (ICMLA'02), June 2002, Monte Carlo Resort, Las Vegas, Nevada, USA

[Harold 1997] Harold Szu, Masud Cader, "Stochastic Neural Networks", Handbook of Neural Computation, IOP Publishing Ltd and Oxford University Press, 1997

[Hassibi 1993] B. Hassibi, D. G. Stock, "Second order derivatives for network pruning; Optimal brain surgeon", Advances in Neural Information Processing Systems 5, Morgan Kaufmann, pp. 164-171, 1993

[Haykin 1994] S. Haykin, "Neural networks: a comprehensive foundations", New York: MacMillian Publishing, 1994

[Hayvarinen 1999] A. Hyvarinen, E. Oja, "Independent Component Analysis: A Tutorial", http://www.cis.hut.fi/aapo/papers/IJCNN99_tutorialweb, 1999

[Hayvarinen 2001] A. Hyvarinen, J. Karhunen, E. Oja, "Independent Component Analysis", Wiley, 2001

[Hayward 1996] R. Hayward, C. Ho-Stuart, J. Diederich, E. Prop, "RULENEG: Extracting rules from a trained ANN by stepwise negation", Neurocomputing Res. Centre, Queensland Univ., Technical Report, 1996

[Hecht 1990] R. Hecht-Nielsen, "Neurocomputing", Adison-Wesley, Reading, MA, 1990

[Hen 2002] Yu Hen Hu, Jenq-Neng Hwang, "Handbook of Neural Network Signal Processing", CRC Press, 2002
[Hermans 1982] J. Hermans et. al., "Manual for the ALLOC80 discriminant analysis program", Leiden, The Netherlands, 1982
[Hinton 1986] G. E. Hinton, J. L. McClelland, D. E. Rumelhart, "Distributed Presentations", Parallel Distributed Processing, vol. 1, pp. 77-109, 1986
[Hoffmann 2002] F. Hoffmann et. al., "Comparing a Genetic Fuzzy and a Neurofuzzy Clasifier for Credit Scoring"
[Horikawa 1993] Y. Horikawa, "Landscapes of Basins of Local Minima in the XOR Problem", International Joint Conference on Neural Networks, vol. 2, pp. 1667-1680, New York, 1993
[Holland 1992] J. Holand, "Adaptation in Natural and Artificial Systems". MIT Press, 1992
[Hollmen 1996] J, Hollmen, "Process Modeling", http://www.cis.hut.fi/~jhollmen
[Holte 1993] R. C. Holte, "Very simple classification rules perform well on most comonly used datasets", Machine Learning, 11, pp. 63-91, 1993
[Ho Tu Bao 2002] HO Tu Bao, "Introduction to Knowledge Discovery and Data Mining", ttp://www.netnam.vn/unescocourse/knowlegde/know_frm.htm
[Hush 1993] D. Hush, B. Horne, "Progress in Supervised Neural Networks", IEEE Signal Processing Magazine, 01/1993, pp. 8-39
[Islam 2003] M. Tanvir Islam, Yoichi Okabe, "Moderatism Based Gradient Learning Rules For Training Multilayer Neural Networks", ICANN 2003, pp. 94-97
[Jacobson] H. Jacobson, "Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review"
[Jain 1998] L. C. Jain, N. M. Martin, "Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications", CRC Press, 1998
[Jankowski 1999] N. Jankowski, "Ontogeniczne Sieci Neuronowe w Zastosowanie do Klasyfikacji Danych Medycznych", PhD thesis, Uniwersytet Mikołaja Kopernika, Toruń 1999
[Jankowski 2003] N. Jankowski, "Ontogeniczne sieci neuronowe. O sieciach zmieniających swoją strukturę", Exit, Warszawa 2003
[Kalman 2001] D. Kalman, "A Singular Valuable Decomposition: The SVD of a Matrix"
[Karbowski 1999] T. Karbowski, "Automatic mammographic screening using artificial neural networks", Proc. 4th Conf. Neural Networks and their Applications, Zakopane 1999
[Karras 2001] D. A Karras, S. A Karkanis, D. K Iakovidis, D. E Maroulis, B. G. Mertzios, "Improved Defect Detection in Manufacturing Using Novel Multidimensional Wavelet Feature Extraction Involving Vector Quantization And PCA Techniques", 8th Panhellenic Conference on Informatics, Nicosia, Cyprus, November 2001
[Kasabov 1996] N. Kasabov, "Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering", MIT Press, 1996
[Kasabov 1999] N. Kasabov, B. Woodford, "Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for Building Adaptive, Intelligent Expert Systems"
[Kavzoglu 1999] T. Kavzoglu, "Pruning artificial neural networks: an example using land cover classification of multi-sensor images", International Journal of Remote Sensing, vol. 20, no. 14, pp. 2787-2803, 1999
[Kegl 2000] B. Kegl, A. Krzyżak, T. Linder, K. Zeger, "Learning and Design of Principal Curves", IEEE Transactions on Pattern Analysis ans Machine Intelligence, vol. 22, no. 2, 2000, pp. 281-297

[Kirkpatrick 1983] S. Kirkpatrick, C. D. Gellat, M. P. Vecchi, "Optimization by Simmulated Annealing", Science, 220, pp. 671-680, 1983

[Kohonen 1984] T. Kohonen, "Self-Organization and Associative Memory", Springer-Verlag, Berlin, 1984

[Kohonen 1990] T. Kohonen, "Statistical pattern recognition revisited", Elsevier Science Publishers, North Holland, 1990

[Kordos 2003a] M. Kordos, W. Duch, "Search-based Training for Logical Rule Extraction by Multilayer Perceptron", Proc. of the Joint Int. Conf. on Artificial Neural Networks (ICANN) and Int. Conf. on Neural Information Processing (ICONIP), Istanbul, June 2003, pp. 86-89

[Kordos 2003b] M. Kordos, W. Duch, "Multilayer Perceptron Trained with Numerical Gradient", Proc. of the Joint Int. Conf. on Artificial Neural Networks (ICANN) and Int. Conf. on Neural Information Processing (ICONIP), Istanbul, June 2003, pp. 106-109

[Kordos 2004a] M. Kordos, W. Duch, "On Some Factors Influencing MLP Error Surface", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004, pp. 217-222

[Kordos 2004b] M. Kordos, W. Duch, "Variable Step Search Algorithm for MLP Training", Proc. of the 8th IASTED Int. Conf. on Artificial Intelligence and Soft Computing, Marbella, Spain, September 2004, pp. 215-220

[Kordos 2004c] M. Kordos, W. Duch, "A Survey of Factors Influencing MLP Error Surface", Control and Cybernetics, vol. 33, no. 4, 2004

[Kordos 2004d] M. Kordos, "Directions in Multilayer Perceptron Weight Space", 4th Warsaw International Seminar on Soft Computing, Warsaw, October 25, 2004

[Kordos 2005] M. Kordos, "Search-based Approach to Multilayer Perceptron Training", Studia Informatica, vol. 26, no. 1 (62), 2005

[Kwaśnicka 2004] H. Kwaśnicka, M. Paradowski, "Selection Pressure and Efficiency of Neural Network Architecture Evolving", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004, pp. 444-449

[Le Cun 1990] Y. Le Cun, John S. Denker, Sara A. Solla, "Optimal Brain Damage", Advances in Neural Information Processing Systems II, 1990

[Le Cun 1991] Y. Le Cun, I. Kanter, S. A. Solla, "Second Order Properties of Error Surfaces: Learning Time and Generalization", Advances in Neural Information Processing Systems, vol. 3, pp. 918-924. San Mateo, CA, Morgan Kaufmann, 1991

[Le Cun 1998] Y. Le Cun et. al., "Efficient Backprop", Neural Networks: Tricks of the Trade, vol. 1524 of Lecture Notes in Computer Science, chapter 1, pp. 9-50, Springer, 1998

[Lee 1993] Lee S., Choi Y., "Unconstrained handwritten zip code recognition". proc. WCNN, Portland, Oregon

[Lehr 1996] M. Lehr, "Scaled Stochastic Methods for Training Neural Networks", PhD Thesis, Stanford University, 1996

[Leino 2004] Antti Leino, "Independent Component Analysis: An Overview", http://www.cs.helsinki.fi/u/salmenki/ lda-seminaari04/ica-paper.pdf, April 2004

[Levin 1994] A. U. Levin et. al., "Fast Pruning Using Principal Components", Advances in Neural Information Processing, vol. 6, 1994

[Lewis 2000] R. J. Lewis, "An Introduction to Classification and Regression Tree (CART) Analysis", Annual Meeting of the Society for Academic Emergency Medicine, San Fransisco, 2000

[Lim 2000] T. S. Lim, W. Y. Loh, Y. S. Shih, "A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms", Machine Learning, 40, pp. 203-228, 2000

[Lisboa 1991] P. J. G. Lisboa, S. J. Perentonis, "Complete Solution of the Local Minima in the XOR Problem", Network: Computation in Neural Systems, vol. 2, no. 1, pp. 119-124, 1991

[Liu 2002] Huan Liu et. al., "Discretization: An Enabling Technique", Data Mining and Knowledge Discovery, 6, pp. 393-423, 2002

[Loh 1988] W. Y. Loh, N. Vanichsetakul, "Tree-structured classification via generalized discriminant analysis (with discussion)", Journal of the American Statistical Association, 83, pp. 715-728, 1988

[Loh 1997] W. Y. Loh, S. Y. Shih, "Split Selection Methods for Classification Trees", Statistica Sinica, 7, pp. 815-840, 1997

[Lozowski 1996a] A. Lozowski, T. J. Cholewo, J. M. Żurada, "Symbolic Rule Representation in Neural Network Models", 2nd Conference on Neural Networks and Their Application, vol. 2, pp. 300-305, Szczyrk 1996

[Lozowski 1996b] A. Lozowski, T. J. Cholewo, J. M. Żurada, "Crisp Rule Extraction from Perceptron Network Classifiers", The IEEE International Conference on Neural Networks", pp. 94-99, Washington, June 1996

[Łęski 2001] J. Łęski, "Ordered Weighted Generalized Conditional Possibilistic Clustering", Zbiory Rozmyte i ich Zastosowanie, Wyd. Politechniki Śląskiej, Gliwice 2001

[Łęski 2002] J. Łęski, N. Henzel, "An ε-insensitive Learning in Neuro-Fuzzy Modeling", 6th Int. Conf. on Neural Networks and Soft Computing, Zakopane, Poland, June 2002, pp. 531-536

[Malheiro 2004] R. Malheiro et. al., "A Prototype for Classification of Classical Music Using Neural Networks", Proc. of The 8th IASTED Int. Conf. on Artificial Intelligence and Soft Computing, Marbella, Spain, September 2004, pp. 294-299

[Mandischer 1993] M. Mandischer, "Representation and Evolution of Neural Networks", Artificial Neural Nets and Genetic Algorithms, pp. 643-649, 1993

[Markowska 2002] U. Markowska-Kaczmar, M. Chumieja, "Rule Extraction From Neural Networks with Evolutionary Algorithms", 6th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2002, pp.370-37

[Markowska 2004] U. Markowska-Kaczmar, P. Wnuk-Lipiński, "Rule Extraction From Neural Networks by Genetic Algorithms with Pareto Optimization", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004, pp. 450-455

[Marks 1999] R. E. Marks, "Genetic Algorithms and Neural Networks: A comparison Based on the Repeated Prisoner's Dilemma", in "Computational Techniques for Modelling Learning in Economics", Springer Verlag 1999

[Marquardt 1963] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", SIAM J. Appl. Math., 1963, vol. 11, pp. 431-441

[Matthews 2000] J. Matthews, "Using Genetic Algorithms with Neural Networks", http:\\www.generation5\org

[McCulloch 1943] W. McCulloch, W. Pitts. "A logical calculus if ideas imminent in nervous activity", Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943

[McKeown 1997] J. J. McKeown, F. Stella, G. Hall, "Some Numerical Aspects of the Training Problem for Feed-forward Neural Nets", Neural Networks, vol. 10(8), pp. 1455-1463, 1997.

[Mertz 1998] C. J. Mertz, C. L. Blake, UCI Repository of Machine Learning Databases, http://www.ics.uci.edu/~mlearn/MLRepository.html

[Michalewicz 2003] Z. Michalewicz, "Algorytmy genetyczne + struktury danych = programy ewolucyjne", WNT, 2003

[Michalski 1986] R. S. Michalski, I. Mozetic, J.Hong, N. Lavrac, "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains", The 5th National Conference on Artificial Intelligence, pp. 1041-1045, Philadelphia, PA, Morgan Kaufmann, 1986

[Michalski 1995] R. S. Michalski, J. Wnek, K. Kaufman, E. Bloedorn, "Inductive learning system AQ15c: The method and user's guide", Reports of the machine learning and Inference Laboratory MLI 95-4, George Mason University, Fairfax, 1995

[Mitra 2002] Sushmite Mitra et. al., "Data Mining in Soft Computing Framework: A Survey", IEEE Transactions on Neural Networks, vol. 13, no. 1, January 2002

[Möller 1993] M. F. Moller, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", Neural Networks, vol. 6, pp. 525-533, 1993

[Mulawka 1997] J. Mulawka, "Systemy ekspertowe", WNT, 1997

[Müller 1997] W. Müller, F. Wysotzki, "The decision-tree algorithm CAL5 based on a statistical approach to its splitting algorithm", Machine Learning and Statistics: The Interface, pp. 45-65, 1997

[Murthy 1997] S. K. Murthy, "On Growing Better Decision Trees from Data", PhD thesis, The John Hopkins University, Baltimore, Maryland, 1997

[Nauck 1996] D. Nauck, R. Kruse, "Designing Neuro-fuzzy Systems through Backpropagation", Fuzzy Modelling: Paradigms and Practice, Kluwer, Boston, pp. 203-228, 1996

[Nauck 1999] D. Nauck, "Design and Implementation of Neuro-Fuzzy Data Analysis Tool in Java", Technische Universitat Brauschweig, 1999

[Naud 2001] A. Naud, "Neural and Statistical Methods for the Visualization of Multidimensional Data", PhD thesis, Uniwersytet Mikołaja Kopernika, Toruń 2001

[Neumann 1998] J. Neumann, "Classification and Evaluation of Algorithms for Rule Extraction from Artificial Neural Networks", PhD Summer Project, University of Edingurgh

[Ng 2004] Sin-Chung Ng, Chi-Chung Cheung, Shu-Hung Leung, "Magnified Gradient Function With Deterministic Weight Modification in Adaptive Learning", IEEE Transactions on Neural Networks, vol. 15, no. 6, November 2004, pp. 1411-1432

[NN Tolbox 2004] Neural Network Toolbox v.4.0.1 User's Guide for Matlab R14, http://www.mathworks.com, 2004

[Nunez 2002] H. Nunez, C, Angulo, A. Catala, "Rule extraction from support vector machines", European Symposium on Artificial Neural Networks, Bruges, Belgium, 2002, pp. 107-112

[Osowski 1996] S. Osowski, "Sieci Neuronowe w Ujęciu Algorytmicznym", WNT, Warszawa 1996

[Palade 2001] V. Palade, D. C. Neagu, R. J. Patton, "Interpretation of Trained Neural Networks by Rule Extraction", International Conference on Computational Intelligence, 7th Fuzzy Days in Dortmund, October 1-3, 2001

[Pennington 2003] M. Pennington, "C4.5 Rule Preceded by an Artificial Neural Network Ensembly for Medical Diagnosis", http://www.dcs.shef.ac.uk/teaching/eproj/ug2003/pdf/u0mp.htm

[Pincho 1993] A. J. Pincho, "Modelling non-linear edge-detectors using artificial neural networks", Int. Conf. IEEE Engineering in Medicine and Biology Society, San Diego, vol. 15, pp. 306-307, 1993

[Press 1992] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, "Numerical Recipes in C", Press Syndicate of The University of Cambridge, 1992, http://www.library.cornell.edu/nr/cbookcpdf.html

[Quinlan 1986] J. R. Quinlan, "Inducion of Decision Trees", Machine Learning 1/1986, pp. 81-106

[Quinlan 1993] J. R. Quinlan, "C4.5: Programs for Machine Learning", Morgan Kaufmann, 1993

[Ranga 2004] N. N. R. Ranga Suri, Dipti Deodhare, P. Nagabhushan, "Parallel Levenberg-Marquardt-based Neural Network Training on Linux Cluster – A Case Study", http://www.ee.iitb.ac.in/~icvgip/PAPERS/248.pdf

[Ranganathan 2004] A. Ranganathan, "The Levenberg-Marquardt Algorithm", http://www.cc.gatech.edu/people/home/ananth

[Riedmiller 1992] M. Riedmiller, H. Braun, "RPROP – a fast adaptive learning algorithm", Technical Report, University Karlsruhe, 1992

[Rivest 2002] F. Rivest, "Knowledge Transfer in Neural Networks: Knowledge-Based Cascade-Correlation", MSc Thesis, MCGill University, Montreal, 2002

[Rosenblatt 1958] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain", Phychological Review, vol. 65, pp. 386-408, 1958

[Raghupathi 1996] Raghupathi, Schkade, Raju, "A Neural Network Approach to Bankruptcy Prediction", NN in Finance and Investing, pp. 227-241, 1996

[Rumelhart 1986] D. E. Rumelhart, J. L. MacClelland, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", vol. 1, MIT Press, Cambridge, 1986

[Rumelhart 1986b] D. E. Rumelhart et al., "Learning Internal Representations by Error Backpropagation", Parallel Distributed Processing, vol. 1, pp. 318-362, MIT Press, Cambridge, MA, 1986

[Rutkowska 1997] D. Rutkowska, M. Piliński, L. Rutkowski, "Sieci neuronowe, algorytmy genetyczne i systemy rozmyte", PWN, Warszawa 1997

[Rutkowski 2003] L. Rutkowski, K. Cpalka, "Flexible Neuro-Fuzzy Systems", IEEE Transactions on Neural Networks, vol. 14, no. 3, pp. 545-574

[Saarinen 1993] S. Saarinen, R. Bramley, G. Cybenko, "Ill-conditioning in Neural Network Training Problems", SIAM Journal of Scientific Computing, vol. 14(3), pp. 693-714, 1993

[Schalkoff 1992] R. Schalkoff, "Pattern Recognition: Statistical, Structural and Neural Approaches", Wiley, 1992

[Schiffmann 1993] W. Schiffmann, M. Joost, R. Werner, "Comparison of Optimized Backpropagation Algorithms", ESANN'93, Brussels, pp. 97-104, 1993

[Schmidt 2002] V. A. Schmidt, C. L. Philip Chen, "Using the Aggregate Feedforward Neural Network for Rule Extraction", International Journal of Fuzzy Systems, vol. 4, no. 3, September 2002

[Setiono 1995] R. Setiono, H. Liu, "Understanding neural networks via rule extraction", the 14th Int. Joint Conf. on Artificial Intelligence, pp. 480-485, Montreal, Canada

[Setiono 2000a] R. Setiono, "Extracting M-of-N rules from trained neural networks", IEEE Transactions on Neural Networks, vol. 11, no. 2, pp. 512-519

[Setiono 2000b] R. Setiono and W.K. Leow, "FERNN: An algorithm for Fast Extraction of Rules from Neural Networks", Journal of Applied Intelligence, vol. 12, no. 1/2, pp. 15-25

[Shang 1996] Y. Shang, B.W. Wah, "Global Optimization for Neural Network Training", IEEE Computer, 29, pp. 45-54, 1996

[Singh 2001] Samer Singh, "Quantifying Structural Time Varying Changes in Helical Data", Neural Computing and Applications, vol. 10, issue 2, pp. 148-154, 2001

[Sordo 2002] M. Sordo, "Introduction to Neural Networks in Healthcare", OpenClinical, 2002

[Spaanenburg 2003] L. Spaanenburg et. al., "Natural learning of neural networks by reconfiguration", SPIE Int. Symp. on Microtechnologies for the new Millennium, Maspalomas, Gran Canaria, Spain, pp. 273-284, 2003

[Statlog 1994] D. Michie, D. J. Spiegelhalter, C. C. Taylor, "Machine Learning, neural and statistical classification", Elis Horwood, London, 1994

[Seiffert 2001] U. Seiffert, "Multiple Layer Perceptron Training Using Genetic Algorithm", Proc. of the 9th European Symposium on Artificial Neural Networks ESANN 2001, Bruges, Belgium, April 25-27, 2001, pp. 159-164, D-Facto, Evere, Belgium, 2001

[Solla 1988] S. A. Solla et. al., "Accelerated Learning in Layered Neural Networks", Complex Systems, vol. 2, pp. 625-640, 1988

[Sontag 1989] E. D. Sontag, H. J. Sussman, "Backpropagation Can Give Rise to Spurious Local Minima Even for Networks Without Hidden Layers", Complex Systems, vol. 3, pp. 91-106, 1989

[Sprinkhuizen 1996] I. G. Sprinkhuizen-Kuyper, E. J. W. Boers, "The error surface of the simplest XOR network has only global minima", Neural Computation, vol. 8, pp. 1301-1320, 1996

[Sprinkhuizen 1998] I. G. Sprinkhuizen-Kuyper, E. J. W. Boers, "The error surface of the 2-2-1 XOR network: the finite stationary points", Neural Networks, vol. 11(4), pp. 683-690, 1998

[Ster 1996] B. Ster, A Dobnikar, "Neural Networks in medical diagnosis: Comparison with other methods", EANN '96, pp. 427-430, 1996

[Sussman 1992] H. J. Sussmann, "Uniqueness of the weights for minimal feedforward nets with a given input-output map", Neural Networks, 5:589-593, 1992

[Taha 1996] I. Taha, J. Gosh, "Three techniques for extracting rules from feedforward networks", Intelligent Engineering Systems Through Artificial Neural Networks, vol. 6, pp. 23-28

[Towell 1991] G. Towell, "Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction", PhD Thesis, University of Wisconsin, Madison

[Thrun 1995] S. Thrun, "Extracting Rules from Artificial Neural Networks with Distributed representation", "Advances in Neural Information Processing Systems", 7, 1995

[Tznankou 2001] E. Micheli-Tzanakou (ed) "Supervised and Unsupervised Pattern Recognition: Feature Extraction and Computational Intelligence", CRC Press 2001

[Ultsch 1995] A. Ultsch. D. Korus, T. O. Kleine, "Integration of Neural Networks and Knowledge-Based Systems in Medicine", 5th Conf. on Artificial Intelligence in Medicine Europe, Pavia, Italy 1995, pp. 425-426

[UMK-KMK] Katedra Metod Komputerowych Uniwersytetu Mikołaja Kopernika w Toruniu, http://www.phys.uni.torun.pl/kmk

[Unnikrishnan 1994] K. P. Unnikrishnan, K. P. Venugopal, "Alopex: A Correlation-Based Learning Algorithm for Feed-Forward and Recurrent Neural Networks", Neural Computations, 6, pp. 469-490, 1994

[Vapnik 1995] V. Vapnik, "The Nature of Statistical Learning Theory", Springer-Verlag, New York, 1995

[Verma 1999] B. Verma, B. Blummenstain, S. Kulkarni, "A New Compression Technique Using an Artificial Neural Network", Journal of Intelligent Systems, 9, pp. 39-53, 1999

[Weigend 1990] A. S. Weigend, D. E. Rumelhart, B. A. Huberman, "Back-propagation, Weight Elimination and Time Series Prediction", Connectionist Model Summer School, Morgan Kaufmann, pp. 65-80, 1990

[Weigend 1991] A. S. Weigend, D. E. Rumelhart, B. A. Huberman, "Generalization by Weight Elimination with Application in Forecasting", Advances in Neural Information Processing Systems, San Mateo, CA, pp. 875-882, 1991

[Weir 2000] M. K. Weir, J. P. Lewis, G. Milligan, "Using Tangent Hyperplanes to Direct Neural Training"

[Weiss 1990] S. M. Weiss, I. Kapouleas, "An empirical comparison of pattern Recognition, neural nets and machine learning classification methods", Reading in Machine Learning, Morgan Kauffman Publ, CA, 1990

[Wejchert 1991] J. Wejchert, G. Tesauro, "Visualizing Processes in Neural Networks", IBM Journal of Research and Developement, 35(1/2), pp. 244-253, 1991

[Werbos 1974] P. Werbos, "Beyond regression: new tools for prediction and analysis in the behavioral science", Doctoral Dissertation, Harward, Cambridge, MA, 1974

[Wilson 2003] D. Randal Wilson, Tony R. Martinez, "The Inefficiency of Batch Training for Gradient Descent Learning", Neural Networks, vol. 16, pp. 1429-1451, 2003

[Yang 2003] Jing Yang et. al., "Visual Hierarchical Dimension Reduction for Exploration of High Dimensional Datasets", Joint Eurographics - IEEE TCVG Symposium on Visualization, 2003

[Yao 2003] J. T. Yao, "Knowledge Based Descriptive Neural Network", http://www2.cs.uregina.ca/~jtyao/Papers/1215.pdf

[Yoon 1994] B. Yoon, R. Lacher, "Extracting rules by destructive learning", The IEEE Int. Conf. on Neural Networks, vol. 3, pp. 1766-71

[Zarate 2004] L. Zarate et. al. "Sensitivity Analysis Obtained Through Artificial Neural Networks – Application in Solar Energy Systems", Proc. of The 8th IASTED Int. Conf. on Artificial Intelligence and Soft Computing, Marbella, Spain, September 2004, pp. 289-293

[Zarndt 1995] F. Zarndt, "A comprehensive case study: An examination of machine learning and connectionists algorithms", MSc Thesis, Department of Computer Science, Brigham Young University, 1995

[Zhang 2000] G. P. Zhang, "Neural Networks for Classification: A Survey", IEEE Transactions on Neural Networks, vol. 30, no. 4, November 2000

[Zhengz 1998] Z. Zheng, G. I. Webb, "Multiply Boosting: A Combination of Boosting and Bagging", The 4th International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 1133-1140, CSREA Press, 1998