Optimization of High Order Perceptrons

Thesis

presented at the

Signal Processing Laboratory

of the

Swiss Federal Institute of Technology (EPFL)

by

Georg Thimm

proposed to the jury:
Prof. M. Kunt
Prof. C. Pelligrini
Prof. W. Gerstner
Prof. L.C. Jain
E. Fiesler (Ph.D.)

Longtemps on a pensé que l'informatique en général et les programmes d'intelligence artificielle en particulier allaient mélanger et présenter sous des angles neufs les concepts humains. Bref, on attendait de l'électronique une nouvelle philosophie. Mais même en la présentant différemment, la matière reste identique : des idées produites par des imaginations humaines. C'est une impasse. La meilleure voie pour renouveler la pensée est de sortir de l'imagination humaine

Bernard Werber
Les fourmis

Version Abrégée

Les réseaux neuronaux sont souvent appliqués en recherche et dans l'industrie. Malheureusement, beaucoup de détails techniques rendent leur application difficile. En particulier, le choix des paramètres d'entraînement et de la topologie du réseau posent un problème. Le but de cette thèse est donc la détermination ou l'élimination des paramètres mentionnés, qui doivent être spécifiés par l'utilisateur. Ensuite, le genre d'applications pour lesquelles les réseaux neuronaux sont utiles sont discutés.

Parmi l'ensemble des paramètres d'entraînement, une attention particulière est donnée au taux d'apprentissage, à la pente des fonctions sigmoidales et à la valeur des poids initiaux. Un théorème permettant l'élimination de ces paramètres est prouvé. En plus, il est démontré que, pour les perceptrons d'ordre supérieur (*high order perceptrons*), des poids initiaux avec de petites valeurs aléatoires sont souvent optimaux du point du vue du temps d'entraînement et de la généralisation.

Le deuxième problème principal est la recherche d'une topologie du réseau qui convient à une application donnée. Pour cette raison, les perceptrons d'ordre supérieur sont préférés à d'autres architectures de réseaux neuronaux, parce qu'ils n'exigent pas de couches de neurones cachés et contournent donc la difficulté de choisir leur nombre et leurs tailles. Par contre, l'ordre et la connectivité du réseau doivent être déterminés. Deux différentes approches permettent cela. La première consiste à élaguer des connexions pendant l'entraînement du réseau de grande taille initiale, l'autre procède en agrandissant peu à peu un réseau d'une petite taille. Les deux genres d'approche sont étudiés, des algorithmes correspondants sont développés et appliqués aux perceptrons d'ordre supérieur. Les (dés-)avantages de ces deux approches et leurs performances sont comparés expérimentalement.

Une perspective des futures recherches sur l'interprétation et l'analyse des perceptrons d'ordre supérieur est ensuite donnée.

Finalement, les perceptrons d'ordre supérieur et les algorithmes développés sont testés sur diverses applications de la vie courante. Les performances obtenues pendant ces expériences sont comparées à celles des autres approches, afin d'en démontrer l'efficacité.

Abstract

Neural networks are widely applied in research and industry. However, their broader application is hampered by various technical details. Among these details are several training parameters and the choice of the topology of the network. The subject of this dissertation is therefore the elimination and determination of usually user specified learning parameters. Furthermore, suitable application domains for neural networks are discussed.

Among all training parameters, special attention is given to the learning rate, the gain of the sigmoidal function, and the initial weight range. A theorem is proven which permits the elimination of one of these parameters. Furthermore, it is shown that for high order perceptrons, very small random initial weights are usually optimal in terms of training time and generalization.

Another important problem in the application of neural networks is to find a network topology that suits a given data set. This favors high order perceptrons over several other neural network architectures, as they do not require layers of hidden neurons. However, the order and the connectivity of a network have to be determined, which is possible by two approaches. The first is to remove connections from an initially big network while training it. The other approach is to increase gradually the network size. Both types of approaches are studied, corresponding algorithms are developed, and applied to high order perceptrons. The (dis-)advantages of both approaches are gone into and their performance experimentally compared.

Then, an outlook on future research on the interpretation and analysis of high order perceptrons and their feasibility is given.

Finally, high order perceptrons and the developed algorithms are applied to a number of real world applications, and, in order to show their efficiency, the obtained performances are compared to those of other approaches.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Why Neural Networks?

A question imposing itself when looking on the various development tools, respectively programming paradigms, as for example object oriented, functional, logic, and constraint logic programming, is, whether neural networks actually add something new and advantageous.

Software development is a very time consuming and expensive process. It is therefore a major aim of research in computer science to cut down the costs of process by trying to automate the software development. Neural network can be used to help this: given some data, the relation between them can be learned by neural networks (more or less) automatically. This, however, is difficult with a system based on, for example, logic or constraint logic programming as it does not match their programming paradigms. A program written under such a paradigm requires that a problem and its solution are describable in some kind of meta-language. Typical tasks, that are difficult to grasp by "conventional" approaches, do not posses a known (mathematically) well-defined relation between the input and the target values, like for example handwritten character recognition (no mathematical function defines the shape of a letter or digit) the only proper reference are prototypes given by humans. A further example is speech recognition, and numerous others exist (see [Fiesler-97], part F).

Another advantage of neural networks is their inherent parallelism which eases an efficient implementation and gives an important gain in computing time. Even though implementations of various programming paradigms promise parallelism, it is usually up to the users to "parallelize" their algorithm which implies additional efforts and problems.

1.2 Where to Apply Neural Networks?

As neural networks have specific strengths, they also have specific problem domains for which they are efficient and useful. On the other hand, for certain applications, neural networks have no advantages over other techniques or even are condemned to fail. An outline of types of problems, for which neural networks are or are not suitable, is therefore important. As neural networks "compete" in this sense with "conventional" algorithms and programming techniques, it is the best to compare them by means of their differences:

Learning It is an outstanding quality of neural networks, that they are able to "learn" from prototypes. This is an important advantage of neural networks over other techniques as it permits to handle problems which are defined only by sets of prototypes.

Generalization Neural networks are able to extrapolate to a certain extent from the training to previously unseen data; a capability which is often called *generalization*[1]. Although the responses of a neural network for untrained data are usually affected by an error, they can be used for applications that do not demand a perfect response. Furthermore, for some problems a unique and correct answer might not even exist.

Uncertainty The exact response of most neural networks can not be predicted for non-examined data, at best a mean difference between the outputs of the network and correct values, respectively the probability for a right classification, can be given. Neural networks are therefore usually excluded from applications requiring a 100% correct response. Such tasks are, for examples, those carrying an important security risk (like in aeronautics [Zijderveld-96]). However, first approaches towards neural networks with a verifiable behavior are done [Wen-96].

Expressiveness In principle, feedforward neural networks like multilayer and high order perceptrons are functions. Therefore they are preferably applied to problems where an *a priori* unknown function has to be approximated. Unknown means that even the type of function is unknown: if only some parameters of a function have to be determined, an approximation of these parameters will result in a solution of a higher quality than obtainable with neural networks, which rules out the latter. This is highly related to the "no free lunch" theorem shown by D. Wolpert which, among other things, implies that an algorithm using *a priori* knowledge will perform better than others [Wolpert-96].

But this is not the only constraint. Practice shows that the function, respectively the border between classes, to be approximated has to be rather "smooth". The direct implementation of functions like data encoding schemes, en-, or decryption as feed forward neural networks are definitively beyond the scope of current technology. Note that they can still be useful in these domains, as shown by J. Schmidhuber and S. Heil, who used neural networks to increase the efficiency of data compression methods like Huffman-coding and arithmetic coding [Schmidhuber-96].

[1] *Performance* is often used as a synonym for generalization. However, in this dissertation it is used in a more general sense.

Complexity The training of neural networks, especially on serial hardware, can be very expensive in terms of computer resources. This might exclude them for applications with real time or hardware constraints. On the other hand, recall can be very fast and parallel hardware easily exploitable.

A "rule of thumb" for the application of neural networks is therefore: *apply neural networks only if the task is basically an approximation of an unknown but "smooth" function or a classification problem with "smooth" class boundaries. Don't use neural networks whenever an effective, conventional algorithm exists or an occasionally "wrong answer" is unacceptable.*

1.3 The Problems of Neural Networks

Theory on neural networks still lacks conclusive answers for many questions. The most important problem is the user-unfriendliness or, how easily users can apply neural networks. This is strongly related to the number of parameters to be determined before a neural network can be successfully trained and integrated it into an application. Consequently, research on neural networks should be aimed at eliminating as much user defined parameters as possible by determining them automatically from the data set or by giving heuristic rules for their determination.

The training of neural network requires several parameters as for example initial weights and learning rate. For the determination of these, for successful training of neural networks crucial, parameters exist heuristic rules, but they do not always yield satisfying results. Furthermore, a neural network topology that suits a specific problem has to be chosen. This is a very crucial point, as a neural network with an inappropriate topology will never perform well, no matter what are the other parameters.

The latter point also concerns the response of a neural network for untrained data. However, generalization is not well-defined and usually measured by different statistical methods like cross-validation, variations of the bootstrap method, sandwich estimator, and so on (see for example [Hertz-91], [Tibshirani-96], and [Efron-95]).

1.4 Why Another Neural Network Topology?

Commonly, the standard perceptron is accepted as the first artificial neural network. Although this network has a major shortcoming, namely its restricted classification and approximation capacity, it served as a base model for further development: M. L. Minsky and S. A. Papert proposed what might be called *high order multilayer perceptron* [Minsky-69]. The widely known *multilayer perceptrons*, as well as *high order perceptrons*, can be regarded as simplifications of this network architecture (see [Hertz-91] for standard and multilayer perceptrons and [Fiesler-97] for high order perceptrons). These neural network architectures have a wider range of possible applications but introduce problems like the choice of a network topology.

In contrast to multilayer perceptrons, the special "add-on" of high order perceptrons to the standard perceptron architecture are not layers of additional neurons but the so called *high order connections*. These connections usually forward the product of two or more inputs towards the sink neuron (see chapter 2.1 for details). This type of neural network has some advantages over multilayer perceptrons: its simpler architecture potentially reduces the number of required training parameters, and they are supposed to learn faster while having a shorter response time (although no comparative study exists).

Almost no conclusive study about parameters settings or the construction of high order perceptrons exist, and the quantity of research performed on this type of neural network is incomparably smaller than for multilayer perceptrons. Therefore this dissertation is mainly dedicated to the high order perceptrons.

1.5 Outline of the Following Chapters

In chapter 2 high order and multilayer perceptrons are defined; moreover, various notations concerning neural networks are thoroughly explained. This chapter includes also an explanation of the schemes and parameters applied during training. Further, the data sets used in the experiments are described.

An important relation between the weights, the learning rate, and the sigmoidal function is proven in chapter 3. Then, implications of this relation on different learning rules are given.

The choice of a neural network simulator is discussed in chapter D. As each application of a neural network has its own specific requirements, several noteworthy features are addressed.

Chapter 4 is dedicated to the optimal setting of three important training parameters: the steepness of the sigmoidal function, the initial weights, and the learning rate. Furthermore, the question for the most suitable activation function is addressed. A large amount of experiments is performed in order to obtain good initial values for these parameters and to find an (almost) optimal method for their determination.

As already indicated in the introduction, the topology of a neural network for a specific application is an important parameter but difficult to determine. There are two major approaches to find such a topology, namely pruning and growing methods. Both of them are examined as it is beforehand not clear which produces better networks. Therefore, in chapter 5 the pruning approach is discussed and several methods compared by means of numerous experiments.

In chapter 6, first an overview on constructive approaches for different types of neural network is given. Then an iteratively growing approach and the reasons why it failed are discussed. This is done in the aim to motivate some of the decisions for a successful approach explained in detail in chapter 7. In the latter chapter a method which constructs networks on the base of a Boolean approximation of the data is presented.

Most neural networks have the disadvantage that the "knowledge" hidden in their structure and the weights is hardly interpretable in terms easily understandable by humans.

Chapter 8 discusses this subject and shows informally that a simple *knowledge extraction* from high order perceptrons can be done easily.

In order to show the efficiency of the developed methods, in chapter 9 they are applied to several, in earlier chapters unemployed, data sets.

Chapter 10 concludes the dissertation and gives an outlook to future research.

1.6 Major Contributions

The author is considering the following items as the major contributions of this work to the research on neural networks:

- High order perceptrons are investigated thoroughly.
- A theorem, that sets into a strong relation the learning rate, the weights, and the steepness of the activation function, is proven and applied. This theorem, which is valid for various types feed-forward neural networks, permits to disregard the influence of one of the three parameters on the learning process.
- A weight initialization scheme for high order perceptrons is shown to be almost optimal in terms of training time and generalization of the trained network.
- Several pruning algorithms are compared for performance in terms of training time and generalization of high order perceptrons.
- A constructive algorithm, which is applicable to several feed-forward networks, is proposed and shown to be efficient for high order perceptrons.
- The efficiency of the developed algorithms is validated for real world applications.

Chapter 2

Network Architecture and Learning Rule

This chapter concerns the terminology and notations used in this dissertation, as well as a proper definition of the neural networks which are subject of this dissertation.

A large variety of terms and notations rise from the many different domains which influence the research in neural networks. Besides some terms proper to the neural network field, others from physics, mathematics, neuro-biology, computer science, *etc.* are employed, and name clashes may cause confusion[1]. Until today, although some expressions and terms are widely known and used, no commonly accepted notation of neural networks exist. However, efforts towards its standardization, as well as a sufficient flexibility, are undertaken [Fiesler-97]. The terminology and notations defined in the following sections are therefore mainly provided for multilayer and high order perceptrons.

2.1 The Architecture of Neural Networks

Basically, backpropagation neural networks consist of (compare figure 2.1):

Neurons which are sometimes also called *artificial neurons*, *neurodes*, or *units*. The latter term is used here for both neurons and connections.

Input neurons are receiving data from the outside world and propagate them to other units in the neural network.

Output neurons are forwarding the processed data to the outside world.

Hidden neurons have no connection to the exterior of the neural network.

[1]For example the expression *non-convergence* has a different meaning in mathematics as compared to neural network

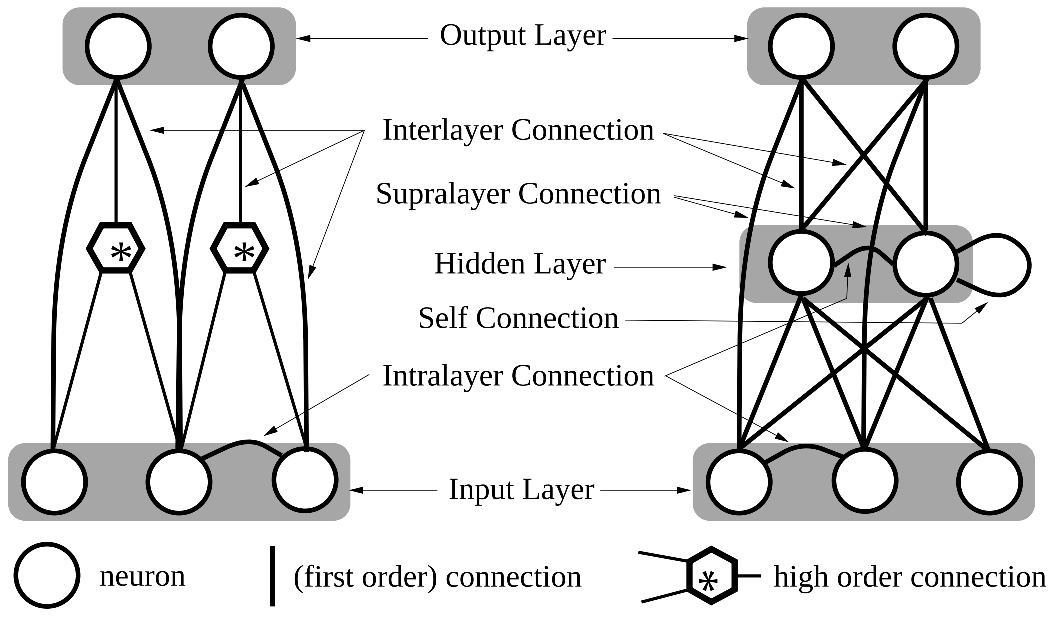


Figure 2.1: The architecture of high order and multilayer perceptrons.

Connections connect the neurons in a neural network: the neurons can be regarded as the processing units and the connections as information transfer lines between them.

Layers can be defined for connections and neurons. However, a consistent definition for all types of neural networks does not exist. As in this work only high order and multilayer perceptrons are examined, the definition can be simplified. In high order perceptrons, all input neurons are exclusively in the first or input layer. Similarly, the output neurons form the output layer. If they exist, as for example in multilayer perceptrons, hidden neurons are grouped into hidden layers, numbered from two up to $L-1$ (L is the number of the output layer, 1 the number of the input layer).

For non-recursive networks without connections among input or output neurons, which is the case for the neural networks examined here, the neurons can be easily assigned to layers: first, the set of hidden neurons is partitioned into a minimal amount of consecutively numbered sets starting with number 2, such that no neuron in one of these sets has a connection going to a neuron in a set of the same or lower index. Then, a hidden neuron is assigned to layer i if it is in partition i.

The term layer may also be used in respect to connections. The enumeration of a connection layer is the same as for the neurons to which they forward their output. Consequently, the smallest number for a connection layer is 2.

Using this definition of layers, different types of connections can be defined for non-recursive networks:

Interlayer connections interconnect neurons in adjacent layers (layer ℓ and ℓ+1). They are the most commonly used type of connections and the only present in standard and multilayer perceptrons.

Intralayer connections which connect neurons in the same layer (these do not appear in non-recursive multilayer perceptrons if the hidden layers are constructed as above),

Self connections connect a neuron to itself (networks with such connections may be considered as being recursive).

Supralayer connections are those connections that do not fit into one of the other categories.

2.2 Notations and Definitions

The notation of the most important parameters of multilayer and high order perceptrons is introduced in this section. Again, these notations are as close as possible to the quasi-standards commonly used (compare also [Fiesler-94] and [Fiesler-97]), with one major exception: usually a connection and its weight are notationally treated as the same entity. This is not done in this dissertation, as the notation of "high order weights" would be confusing.

L is the number of layers of neurons in a neural network. The lowest or input layer has the index 1, layer L is the output layer of the network, and intermediate or hidden layers ℓ are numbered consecutively ($1 \leq \ell \leq L$), if they exist.

N_ℓ is the number of neurons in layer ℓ, with its neurons numbered from 1 up to N_ℓ.

C_ℓ is the number of (high order) connections in layer ℓ of a (high order) multilayer perceptron. It equals the number of weights W_ℓ in the same layer.

$c^p_{\ell,i,\{(\ell_1,n_1),(\ell_2,n_2),\ldots\}}$ is the output of the ith high order connection in a layer ℓ if pattern number p was fed into the network. The usually omitted set of neurons $\{(\ell_1, n_1), (\ell_2, n_2), \ldots\}$ determines from which neurons the output is used as input to this connection (see figure 2.2). Each pair (l, n) represents one input of the connection: n is the number of the neuron in layer l. In the following, the output of a connection is the product of its inputs, but other operations are imaginable and used.

1. Yoh-Han Pao defines the functional link neural networks. In this type of network, each connection applies a function (for example a $sin(ix)$ with $i \in I$) to its single input [Pao-89][Pao-95].
2. P. Liang and N. Jamali introduce quasi polynomials, that are polynomials with non-integer exponents [Liang-93].
3. J. Ghosh and Y. Shin use ridge polynomials [Ghosh-92] [Shin-95].

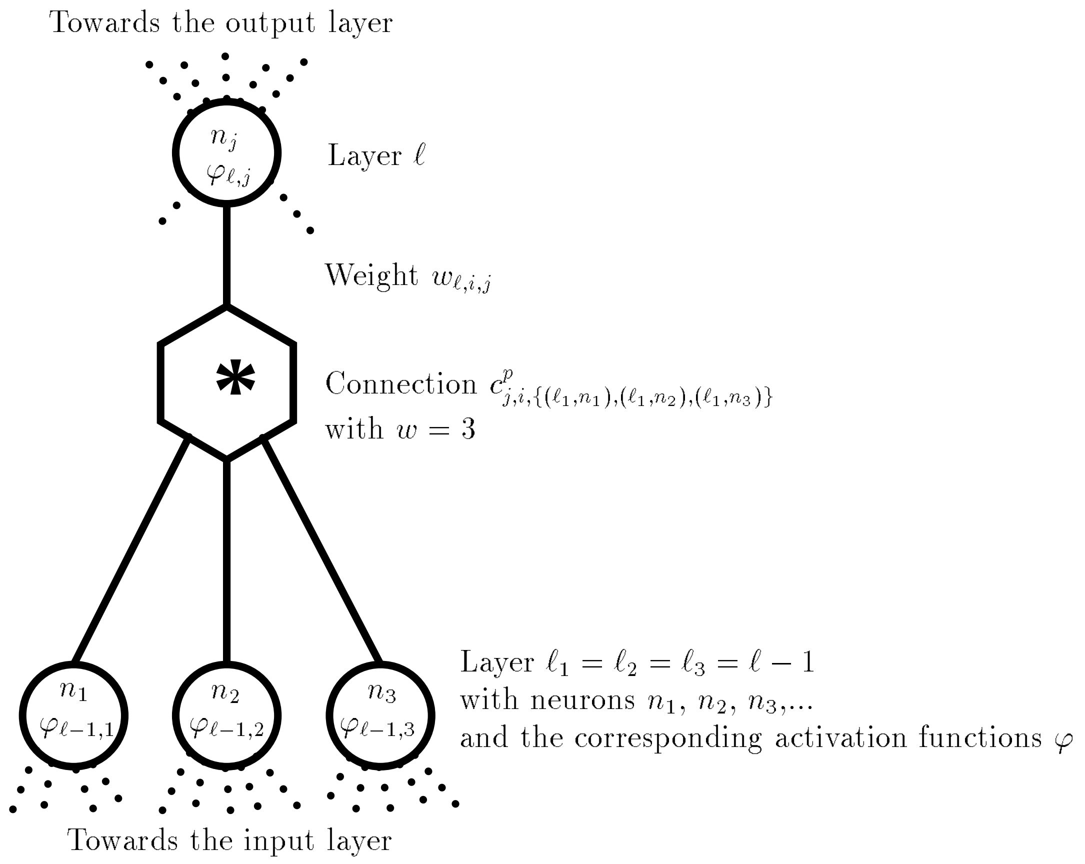

Towards the output layer
n_j
$\varphi_{\ell,j}$
Layer ℓ
Weight $w_{\ell,i,j}$
Connection $c^p_{j,i,\{(\ell_1,n_1),(\ell_1,n_2),(\ell_1,n_3)\}}$
with $w = 3$
n_1
$\varphi_{\ell-1,1}$
n_2
$\varphi_{\ell-1,2}$
n_3
$\varphi_{\ell-1,3}$
Layer $\ell_1 = \ell_2 = \ell_3 = \ell - 1$
with neurons n_1, n_2, n_3,...
and the corresponding activation functions φ
Towards the input layer

Figure 2.2: The notation of high order connection and neurons.

ω The *order of a connection* is defined as the cardinality of the set of neurons $\{(\ell_1, n_1), (\ell_2, n_2), \ldots\}$; the order Ω of a network is defined by the maximal order of a connection in the network.

$w_{\ell,i,j}$ is the weight multiplied with the output of connection i of layer ℓ (see figure 2.2). Neuron j in layer ℓ sums over all these products (the value of this sum is denoted as $h_{\ell,j}$) and applies a so called *activation function* $\varphi()$.

W_ℓ is the number of weights in layer ℓ (compare C_ℓ).

$\varphi_{\ell,j}()$ Mainly four different types of activation functions are used:

- The *identity* (linear) function $id()$.
- The *logistic* function

$$f_\beta(x) = \frac{1}{1 + \exp(-\beta_{\ell,j} x)}. \tag{2.1}$$

- The hyperbolic tangent *tanh*() function. Note that
$$tanh(x) = 2f_2(x) - 1.$$
- The hard limiting threshold or Heavieside function
$$threshold_t(x) = \begin{cases} 0 & \text{if } x \leq t \\ 1 & \text{if } x > t \end{cases} \tag{2.2}$$
is often applied in conjunction with data sets with Boolean target values. The first derivative of this function does not exist which makes it unsuitable for the backpropagation algorithm. Therefore, either the perceptron learning algorithm is applied (see [Hertz-91]), or the error used for the weight update is calculated for another sigmoidal function, and the threshold function is only used during the recall phase. The hyperbolic tangent or the logistic function are common choices as
$$\lim_{\beta \to \infty} f_\beta(x) = threshold_0(x)$$
and
$$\lim_{\beta \to \infty} tanh_\beta(x) = 2 * threshold_0(x) - 1.$$
- The Gaussian activation function which is often called *radial basis function.*

Here only the *linear*, the *logistic*, and the *hyperbolic tangent* are used. Further details on the usage of these activation functions are given and discussed in section 4.1.

$\beta_{\ell,j}$ This parameter is the so called *gain* (also *steepness* in some publications). $\beta_{\ell,j}$ is usually omitted if equal to one.

$a^p_{\ell,j}$ is the activation value of neuron j in layer ℓ ($j > 0$) after the pattern p was propagated trough the network. For $l = L$, this variable is also called the *network output* O^p_j.

t^p_j denotes the target value of pattern p for output neuron j. The value of p ranges between 1 and the number P of patterns.

ξ^p_i is the ith element of input pattern p.

$\eta_{\ell,i,j}$ is the *learning rate* used for the weight with the same indexes. If the whole network uses a unique, *global learning rate*, the indexes are omitted.

$E(\mathbf{W})$ is the *error function*, *objective function*, or *cost function* for a set of target vectors $\mathbf{t}^p$ and the output vectors O^p of the network with a certain instantiation of its weights. The so called *summed squared error* is most often used. It is defined as the sum of squared differences of t_j and O^p_i:
$$E(\mathbf{W}) = \frac{1}{2} \sum_{p,j} (\mathbf{t}^p_j - O^p_j)^2 \tag{2.3}$$

but different variations as for example the cross entropy error are known (see [Miller-91] for a discussion on restrictions for prospective error functions and further references). This includes in the section 5 discussed penalty terms and application related modifications which are beyond the scope of this dissertation.

$\Delta w_{\ell,i,j}$ is the amount by which a weight is changed and is usually calculated from the first derivative of E with respect to weight $w_{\ell,i,j}$ multiplied by the learning rate (see below for details).

In order to simplify the notation the following convention is used: $a_{\ell,0} = 1$ for $1 \leq l < L$. The term *bias* (or *offset*) refers to (the value of) the weights $w_{\ell,0,j}$. This implies that all input vectors to the network are extended by a constant element with the value 1 ($\xi_0 = 1$).

In the following only two special types of networks are examined (compare section 2.1):

Multilayer perceptrons which have a number of hidden layers but neither intra-, supra-, nor high order connections.

High order perceptrons which have neither hidden layers ($L = 2$), nor intra-layer connections, but high order connections.

2.3 The Backpropagation Algorithm

This section defines the neural network models, that are subject of this dissertation, in more detail. Furthermore, a generalization of the on-line backpropagation learning rule [Rumelhart-86] is described in which every neuron has its own local learning rate and gain. The standard case of a unique learning rate corresponds to all local learning rates being equal for the whole network.

The backpropagation algorithm consists of the following six steps:

1. **Initialization** Weights and biases are initialized with random values. Details on this subject can be found in chapter 4[2].

2. **Pattern presentation** The input vector ξ^p of a pattern p is used to initialize the activation values of the neurons in the input layer, and its corresponding target vector $\mathbf{t}^p$ are presented.

$$a_{1,i} := \xi_i^p$$

3. **Forward propagation** During this phase, the activation values of the neurons are propagated layer-wise through the network, starting at the input layer. The activation value $a_{\ell,j}$ of neuron j in layer ℓ ($2 \leq \ell \leq L$) is

[2]See also [Thimm-94.2] for an in-depth study of multilayer perceptron weight initialization.

$$
\begin{aligned}
a_{\ell,j} &= \varphi_{\ell,j}(\beta_{\ell,j} h_{\ell,j}) \text{ with} && (2.4) \\
h_{\ell,j} &= \left(\sum_{i=0}^{N_\ell} w_{\ell,i,j} \; c_{\ell,i,\{(\ell_1,n_1),(\ell_2,n_2),\ldots\}} \right) && (2.5)
\end{aligned}
$$

where $\varphi_{\ell,j}$ is a differentiable activation function, for example the logistic function (see section 2.2). The output of the connections is (assuming the multiplication is the splicing function):

$$c_{\ell,i,\{(\ell_1,n_1),(\ell_2,n_2),\ldots\}} = \prod_{(\ell',n')\in\{(\ell_1,n_1),(\ell_2,n_2),\ldots\}} a_{\ell',n'} \tag{2.6}$$

4. **Backward propagation** In this phase the weight changes $\Delta w_{\ell,i,j}$ are calculated from the derivative of the error function to the weights:

$$\Delta w_{\ell,i,j} = \eta \frac{\partial E}{\partial w_{\ell,i,j}}.$$

The calculation of the $\Delta w_{\ell,i,j}$ can be simplified for multilayer perceptrons with no supra- nor intralayer connections and for high order perceptrons. An efficient algorithm for the calculation of the Δw is formulated below. This algorithm assumes that all connections except those which receive input from the input layer are of first order, and only interlayer connections are present. The latter is true for standard multilayer and high order perceptrons. For each neuron an error signal δ is calculated, starting at the output layer, and then propagated back through the network towards the input layer (the superscripts p for the pattern number are omitted):

$$
\begin{aligned}
\delta_{L,j} &= \beta_{L,j}\varphi'_{L,j}(\beta_{L,j}h_{L,j})\,(t_j - O_j) && \text{for the output layer } L \\
\delta_{\ell,j} &= \beta_{\ell,j}\varphi'_{\ell,j}(\beta_{\ell,j}h_{\ell,j})\,\textstyle\sum_k \delta_{\ell+1,k} w_{\ell+1,j,k} && \text{for layers } 2 \leq \ell \leq L-1.
\end{aligned} \tag{2.7}
$$

5. **Weight update** After the calculation of all error signals, the weights and biases are updated on-line:

$$w_{\ell,i,j} := w_{\ell,i,j} + \eta_{\ell,j}\delta_{\ell,j}c_{\ell-1,i}, \tag{2.8}$$

where $\eta_{\ell,j}$ denotes the learning rate of neuron j in layer ℓ.

Usually, the weight update is done *on-line*, that is after each pattern presentation. The counterpart to this is the *off-line backpropagation* where all Δw are summed over the whole pattern set and afterwards applied to the neural network.

5. **Convergence test** This step is performed after all training patterns are trained once, and the weights are updated. If the mean square error for the training data and the network output is too big (the network did not converge) go to back to the pattern presentation in step **2.**. This step is usually performed after each pattern is presented once (one sweep over the pattern set was done).

2.4 Used Data Sets

In the following chapters, various algorithms for the optimization of the generalization, convergence time, and size of neural networks are compared. Usually, the only way to do so is to train networks on different data sets, as the mathematics of neural networks are not (yet?) enough developed.

Most of the data sets used during such comparisons are obtained (if not stated otherwise) from an anonymous-ftp server at the University of California [Murphy-96] which also contains further references and documentation. Each entry in the list below is preceeded with the name of the data set used in this dissertation. The names are followed by number of input and output elements, the size of the training and test set.

CES (inputs: 2, outputs: 1, train: 20, test: 10) is the output of the constant elasticity of a substitution production function for thirty pairs of labour and capital input (see [Judge-85], pages 195 and 210). The patterns have two real valued inputs and one real valued target, none of them scaled.

Auto-mpg (inputs: 7, outputs: 1, train: 294, test: 98) concerns the city-cycle fuel consumption of cars in miles per gallon, to be predicted in terms of 3 multi-valued discrete and 4 continuous attributes. All values are scaled to the interval $[0, 1]$ (incomplete patterns have been removed). For further documentation see[Quinlan-93].

Solar (inputs: 12, outputs: 1, train: 209, test: 59) contains the sun spot activity for the years 1700 to 1990. The task is to predict the sun spot activity for one of those years, given the activity of the preceding twelve years ($\hat{=}$12 real valued inputs). The data are scaled to the interval $[0, 1]$.

Servo (inputs: 12, outputs: 1, train: 84, test: 83) was created by Karl Ulrich (MIT) in 1986 and contains a very non-linear phenomenon: predicting the rise time of a servomechanism in terms of two (continuous) gain settings and two (discrete) choices of mechanical linkages. The input is coded into two groups of five Boolean and two discrete values, one assuming four, the other five values. The target is real valued and, like all real valued inputs, scaled to the interval $[0, 1]$.

Glass (inputs: 9, outputs: 1, train: 164, test: 50) consists of eight scaled weight percentages of certain oxides and a 7 valued code for the type of glass (window glass, head lamps, *etc.*). The target is the refractive index of the glass, scaled to $[0, 1]$.

British Vowels (inputs: 10, outputs: 11, train: 528, test: 463) is a set of 991 British vowels of different speakers. The inputs are 10 linear prediction coefficients derived log area ratios. The Boolean target values are encoded as $+1$ and -1 [Deterding-89].

Wine (inputs: 13, outputs: 3, train: 137, test: 41) is the result of a chemical analysis of wines grown in a region in Italy derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. A wine has to be classified using these values which are scaled to the interval $[0, 1]$. The target patterns use Boolean values, encoded as $+1$ and -1.

Monks data sets (inputs: 17, outputs: 1, test: 432) These three classification problems presented in [Thrun-91] are defined for the same domain consisting of two Boolean, three 3-valued and a 4-valued attribute. The 3- and 4-valued attributes are coded as Boolean vectors (if the attribute has value n, the n-th Boolean vector element is exclusively true). The classification associated to the Monk's problems are:

Monk 1 (train: 124) is defined as $(a1 = a2)$ *or* $(a5 = 1)$.

Monk 2 (train: 169) is defined as *exactly two of* $\{a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1, a6 = 1\}$.

Monk 3 (train: 122) is defined as $(a5 = 3$ *and* $a4 = 1)$ *or* $(a5 \neq 4$ *and* $a2 \neq 3)$ with 5% class noise added to the training set.

Remark: the test sets of these data sets include also the training data. This is done in order to be conform with the data specification found in [Murphy-96].

Finish Vowels (inputs: 20, outputs: 5, train: 211, test: 89) is a subset of 300 patterns of a vowels data set, obtainable via ftp from cochlea.hut.fu (130.233.168.48) with the LVQ-package (lvq_pak). An input pattern consists of 20 unscaled cepstral coefficients obtained from continuous Finnish speech. The task is to determine whether the pronounced phoneme is a vowel and, in the case it is, which of the five possible ones. The Boolean target values are encoded as $+1$ and -1.

Digits (inputs: 64, outputs: 10, train: 500, test: 500) consists of handwritten digits (50 patterns for each of the ten digits) extracted from the *NIST Special Database 3* [Garris-92]. Each digit was scaled to fit into an image of 8x8 points, and each pixel is represented by an eight bit value. The input values are scaled to the interval $[0, 1]$, and the Boolean target values are encoded as $+1$ and -1.

Chapter 3

A Relation Between Gain, Learning Rate, and Weights

As it will be discussed in chapter 4, the initial weights, the learning rate, and the gains of the sigmoidal functions have a major influence on convergence time and generalization of a feedforward neural network. It is often neglected that optimal values for one these parameters dependent on the value of the other two, as well as they may change with other network or training parameters. Algorithms for the choice of, for example, the learning rate assume usually a standard value of 1.0 for the gain of the activation function (even if not clearly stated). This can be a problem, as certain hardware implementations give constraints for one of these parameters.

I was able to prove the mentioned relation, which gives a good insight into the interaction of these parameters and permits to handle them with more expertise.

3.1 An Equivalence of Neural Networks

The theorem formulated below gives a precise relationship between gain, initial weights, and learning rate for two neural networks trained with the backpropagation algorithm. The theorem states that, if this relationship is fulfilled, the two networks converge in an equal number of training cycles and realize the same function. They are therefore called *equivalent*.

The theorem requires that the two neural networks, which are compared, have an identical topology and corresponding neurons with alike activation functions φ:

$$\forall x \in \Re: \; \varphi_{\ell,j}(\beta_{\ell,j} x) = \widehat{\varphi}_{\ell,j}(\widehat{\beta}_{\ell,j} x). \tag{3.1}$$

In words, corresponding neurons in the two networks have the same activation functions with different gains.

Theorem 1: Two neural networks N and $\widehat{N}$ of identical topology whose activation function pairwise satisfy equation 3.1, and furthermore the gains $\beta_{l,j}$ and $\widehat{\beta}_{l,j}$, learning rate $\eta_{\ell,i,j}$ and $\widehat{\eta}_{\ell,i,j}$, and weights $w_{\ell,i,j}$ and $\widehat{w}_{\ell,i,j}$ fulfill

$$\frac{\beta_{l,j}}{\widehat{\beta}_{l,j}} = \sqrt{\frac{\widehat{\eta}_{\ell,i,j}}{\eta_{\ell,i,j}}} = \frac{\widehat{w}_{\ell,i,j}}{w_{\ell,i,j}}, \tag{3.2}$$

are equivalent under the on-line backpropagation algorithm; that is, their outputs equal each other, when the same pattern set is presented in the same order.

In other words, the gain of the activation function, the weights, and the learning rate in backpropagation neural networks are exchangeable in a certain sense: if one of the parameters is changed and the other are adapted according to equation 3.2, then the behavior of the neural network is unchanged. Note that, albeit the theorem is formulated for any weights, the random case is of primary interest and subject to a more thorough discussion in chapter 4.

Proof: To simplify the notation, the vector of incoming weights of neuron j is denoted by $\mathbf{w}_{\ell,j}$ and the vector of output values of connection layer ℓ by $\mathbf{c}_\ell$. Furthermore, it is assumed that the networks have only first order connections with the exception of the input layer, and that the learning rates and gains of the activation functions are global. Note that the theorem is valid for the non-restricted case, but the proof would be much more difficult due to indexing and notational problems. The variables of network $\widehat{N}$ are distinguished from the variables in network N by an adding " $\widehat{\ }$ "; like for example $\widehat{h}_{\ell,j}$. Using this notation, equation 2.5 can be rewritten as:

$$h_{\ell,j} = \mathbf{w}_{\ell,j} \cdot \mathbf{c}_{\ell-1}, \tag{3.3}$$

where " $\cdot$ " is the inner product operator.

The proof of theorem 1 is separated into two parts, each formulated as a Lemma. The first Lemma deals with the forward propagation and basically states that, under the condition given in the theorem, the networks have the same output for the same input. The second Lemma extends this to the backward propagation and states that the weight changes do not disturb this property.

Lemma 1 Two networks N and $\widehat{N}$, satisfying the preconditions given in theorem 1, have the same activation values for corresponding neurons. More precisely, for all layers $\ell \geq 2$ the equation $\mathbf{a}_\ell = \widehat{\mathbf{a}}_\ell$ is fulfilled if $\mathbf{a}_1 = \widehat{\mathbf{a}}_1$.

Proof Lemma 1 is shown by induction on the number of layers, starting at the input layer.

Induction base: Since the input patterns is fed into the network, the activation values of the input layer neurons of network N and $\widehat{N}$ are identical ($\mathbf{a}_1 = \widehat{\mathbf{a}}_1$), the output of the connections $\mathbf{c}_2 = \widehat{\mathbf{c}}_2$ in the first connection layer yield the same values.

Induction step: For neuron j, not in the input layer:

$$\begin{array}{rrcll}
 & a_{\ell,j} & = & \widehat{a}_{\ell,j} & \text{using (2.4), trivially fulfilled for } j=0. \\
\Leftrightarrow & \varphi_{\ell,j}(\beta_{\ell,j} h_{\ell,j}) & = & \varphi_{\ell,j}(\widehat{\beta}_{\ell,j} \widehat{h}_{\ell,j}) & \text{using (3.1) and (2.5)} \\
\Leftarrow & \beta_{\ell,j} \mathbf{w}_{\ell,j} \cdot \mathbf{c}_{\ell-1} & = & \widehat{\beta}_{\ell,j} \widehat{\mathbf{w}}_{\ell,j} \cdot \widehat{\mathbf{c}}_{\ell-1} & \text{using the induction hypothesis} \\
\Leftrightarrow & \beta_{\ell,j} \mathbf{w}_{\ell,j} \cdot \mathbf{c}_{\ell-1} & = & \widehat{\beta}_{\ell,j} \widehat{\mathbf{w}}_{\ell,j} \cdot \mathbf{c}_{\ell-1} & \text{as the } c_{\ell,i} \text{ are independent} \\
\Leftrightarrow & \beta_{\ell,j} \mathbf{w}_{\ell,j} & = & \widehat{\beta}_{\ell,j} \widehat{\mathbf{w}}_{\ell,j} &
\end{array}$$

which is true on account of equation 3.1 on page 17. ***q.e.d.***

In the proof of Lemma 1 the property $\beta_{\ell,j} \mathbf{w}_{\ell,j} = \widehat{\beta}_{\ell,j} \widehat{\mathbf{w}}_{\ell,j}$ is used. Since the backward propagation changes the weights, it has to be shown that this property is an invariant of the backward propagation step.

Lemma 2 Consider networks N and $\widehat{N}$, with $a_{\ell,j} = \widehat{a}_{\ell,j}$ and $\beta_{\ell,j} h_{\ell,j} = \widehat{\beta}_{\ell,j} \widehat{h}_{\ell,j}$ (for all ℓ and j), then

$$\forall j, \ell : \beta_{\ell,j} \mathbf{w}_{\ell,j} = \widehat{\beta}_{\ell,j} \widehat{\mathbf{w}}_{\ell,j} \tag{3.4}$$

is invariant under the backward propagation step (if the same input and target patterns are propagated through the networks).

Proof Let $\Delta \mathbf{w}_{\ell,j}$ denote the weight change $\eta_{\ell,j} \delta_{\ell,j} \mathbf{a}_{\ell-1}$. It is clear that equation 3.4 holds if and only if $\beta_{\ell,j} \Delta \mathbf{w}_{\ell,j} = \widehat{\beta}_{\ell,j} \Delta \widehat{\mathbf{w}}_{\ell,j}$ (for all j and ℓ). Manipulating this expression:

$$\begin{array}{rrcll}
 & \beta_{\ell,j} \Delta \mathbf{w}_{\ell,j} & = & \widehat{\beta}_{\ell,j} \Delta \widehat{\mathbf{w}}_{\ell,j} & \\
\Leftrightarrow & \beta_{\ell,j} \eta_{\ell,j} \delta_{\ell,j} \mathbf{a}_{\ell-1} & = & \widehat{\beta}_{\ell,j} \widehat{\eta}_{\ell,j} \widehat{\delta}_{\ell,j} \widehat{\mathbf{a}}_{\ell-1} & \text{definition of } \Delta \mathbf{w}_{\ell,j} \\
\Leftrightarrow & \beta_{\ell,j} \eta_{\ell,j} \delta_{\ell,j} & = & \widehat{\beta}_{\ell,j} \widehat{\eta}_{\ell,j} \widehat{\delta}_{\ell,j} & \text{since } \forall \ell : \ \widehat{\mathbf{a}}_\ell = \mathbf{a}_\ell \text{ (lemma 1)}
\end{array}$$

which is shown by an induction on the number of layers, starting at the output layer.

Induction base: For the activation values of the output layer this equation holds:

$$\begin{array}{rll}
& \beta_{L,j}\eta_{L,j}\delta_{L,j} = \widehat{\beta}_{L,j}\widehat{\eta}_{L,j}\widehat{\delta}_{L,j} & \text{using (2.7)} \\
\Leftrightarrow & \beta^2_{L,j}\eta_{L,j}\varphi'_{L,j}(\beta_{L,j}h_{L,j})\,(t_j - a_{L,j}) & \\
& = \widehat{\beta}^2_{L,j}\widehat{\eta}_{L,j}\varphi'_{L,j}(\widehat{\beta}_{L,j}\widehat{h}_{L,j})\,(t_j - \widehat{a}_{L,j}) & \text{since } a_{L,j} = \widehat{a}_{L,j} \\
\Leftarrow & \beta^2_{L,j}\eta_{L,j}\varphi'_{L,j}(\beta_{L,j}h_{L,j}) = \widehat{\beta}^2_{L,j}\widehat{\eta}_{L,j}\varphi'_{L,j}(\widehat{\beta}_{L,j}\widehat{h}_{L,j}) & \text{using (3.2)} \\
\Leftarrow & \varphi'_{L,j}(\beta_{L,j}h_{L,j}) = \varphi'_{L,j}(\widehat{\beta}_{L,j}\widehat{h}_{L,j}), &
\end{array}$$

which follows from applying the chain rule to $\varphi_{L,j}(\beta_{L,j}h_{L,j}) = \varphi_{L,j}(\widehat{\beta}_{L,j}\widehat{h}_{L,j})$, shown in Lemma 1.

Induction step: For a neuron j not in the output layer ($\ell < L$):

$$\begin{array}{rll}
& \beta_{\ell,j}\eta_{\ell,j}\delta_{\ell,j} = \widehat{\beta}_{\ell,j}\widehat{\eta}_{\ell,j}\widehat{\delta}_{\ell,j} & \text{using (2.7)} \\
\Leftrightarrow & \beta^2_{\ell,j}\eta_{\ell,j}\varphi'_{\ell,j}(\beta_{\ell,j}h_{\ell,j})\ \sum_k \delta_{\ell+1,k}\beta_{\ell+1,j}w_{\ell+1,j,k} & \text{using } \varphi_{\ell,j}(\beta_{\ell,j}h_{\ell,j}) \\
& = \widehat{\beta}^2_{\ell,j}\widehat{\eta}_{\ell,j}\varphi'_{\ell,j}(\widehat{\beta}_{\ell,j}\widehat{h}_{\ell,j})\ \sum_k \widehat{\delta}_{\ell+1,k}\widehat{\beta}_{\ell+1,j}\widehat{w}_{\ell+1,j,k} & \quad = \varphi_{\ell,j}(\widehat{\beta}_{\ell,j}\widehat{h}_{\ell,j}) \\
\Leftrightarrow & \beta^2_{\ell,j}\eta_{\ell,j}\ \sum_k \delta_{\ell+1,k}\beta_{\ell+1,j}w_{\ell+1,j,k} & \\
& = \widehat{\beta}^2_{\ell,j}\widehat{\eta}_{\ell,j}\ \sum_k \widehat{\delta}_{\ell+1,k}\widehat{\beta}_{\ell+1,j}\widehat{w}_{\ell+1,j,k} & \text{using (3.2)} \\
\Leftrightarrow & \sum_k \delta_{\ell+1,k}\beta_{\ell+1,j}w_{\ell+1,j,k} = \sum_k \widehat{\delta}_{\ell+1,k}\widehat{\beta}_{\ell+1,j}\widehat{w}_{\ell+1,j,k} & \text{using (3.2)} \\
\Leftrightarrow & \sum_k \delta_{\ell+1,k}\beta_{\ell+1,j}w_{\ell+1,j,k} = \sum_k \widehat{\delta}_{\ell+1,k}\beta_{\ell+1,j}w_{\ell+1,j,k}, &
\end{array}$$

which implies the induction hypothesis $\delta_{\ell+1,j} = \widehat{\delta}_{\ell+1,j}$, as this equation has to be fulfilled for all $\beta_{\ell+1,j}w_{\ell+1,j,k}$. ***q.e.d.***

An induction over the number of pattern presentations, using these lemmas, concludes the proof of theorem 1. ***q.e.d.***

3.2 Extensions and Applications of Theorem 1

For practical applications, the standard backpropagation algorithm as presented in section 2.3 on page 12 is rarely employed. More often, variations are used in order to increase the convergence speed, or constraints given by a certain hardware implementation have to be respected. Although these special cases of the backpropagation algorithm are not covered by Theorem 1, it can be easily extended to several variations of the backpropagation learning rule (such as momentum, flat spot elimination, *etc.*). Furthermore, it can be used to adapt these parameters to hardware given constraints, so that an efficient training of these neural networks is possible. Additionally, similarities between some learning rules can be shown. A summary of these facts is given in table 3.1. The corresponding proofs are omitted, as they are analogous to the proof of theorem 1.

Variation / Modification	Remarks
Batch or off-line learning. Momentum [Rumelhart-86]	Theorem 1 holds without modification of the network parameters.
Flat spot elimination [Fahlman-88]	Theorem 1 holds if the constant $\widehat{c}$, which is added to the derivative when the weight changes are calculated, is adapted in a way that $$\frac{\widehat{c}}{c} = \frac{\widehat{\beta}}{\beta}.$$
Weight discretization with multiple thresholding of the real-valued weights [Fiesler-94]	The thresholds need adaptation. Supposed that d and $\widehat{d}$ are the discretization functions applied on the weights, theorem 1 holds if $\forall x : \beta_{\ell,j} d(x) = \widehat{d}(\beta_{\ell,j} x)$.
Adaptive gain [Plaut-86] [Bachmann-90] [Kruschke-91] [Sperduti-93]	An adaptation $\Delta\beta_{\ell,j}$ of the gain can be replaced by a changing the learning rates from $\beta_{\ell,j}^2 \eta_{\ell,j}$ to $(\beta_{\ell,j} + \Delta\beta_{\ell,j})^2 \eta_{\ell,j}$ and of the weights from $\beta_{\ell,j} \mathbf{w}_{\ell,j}$ to $(\beta_{\ell,j} + \Delta\beta_{\ell,j}) \mathbf{w}_{\ell,j}$.
Adaptive learning rates [Darken-92]	Instead, the weights and gain can be modified.
Faster convergence by using steeper activation functions [Izui-90] [Cho-91]	Is almost equivalent to using a higher learning rate and a bigger weight range. This approach is therefore only efficient if the other two parameters are suboptimal.
Hardware implementation with non-standard gain [Saxena-94]	The theorem can be applied directly to change the initial weight variance and learning rate [Thimm-96].
Increasing gain to obtain a hard limiting threshold [Corwin-94] [Yu-94]	Alternatively, the weights can be multiplied with a constant > 1, and, if this does not cause a degradation in performance, in the final stage of the training process the activation functions are replaced by thresholds.

Table 3.1: Theorem 1 applied to variations and modifications of the backpropagation algorithm.

Addendum

For completeness the author would like to include the reference to a letter in Neural Networks [Jia-94] that was brought to his attention after the submission of the paper [Thimm-96], in which a similar theorem is presented, albeit without proof or applications.

Chapter 4

Optimal Setting of Weights, Learning Rate, and Gain

The training time of the backpropagation rule applied to neural networks depends much on the initial values of the weights and biases, the learning rate(s), the type of sigmoidal function(s), the network topology, and on learning rule improvements like a momentum term, *etc.* The optimal values for these parameters are *a priori* unknown because they depend on the training data set used. In practice it does not make sense to perform a global search for obtaining the optimal values of these parameters, as this requires a huge amount of simulations which means that the task is solved many times before an optimal choice of parameters is found. However, current mathematical techniques are insufficient for a complete theoretical study of the learning behavior of neural networks. Furthermore, it is observable that the convergence time of the network can change significantly for small changes in the initial weights, as was demonstrated by J. F. Kolen and J. B. Pollack [Kolen-90]. It is therefore important to have a good approximation of the optimal initial value of these parameters; or with the words of J. F. Kolen and J. B. Pollack: to start the learning process in the "eye of the storm," to reduce the required training time. As an extensive search for the optimum values can usually not be performed during a practical application of neural networks; the only hope is to find a heuristic for the determination of an almost optimal setting of these parameters.

It can be easily seen that an infinite number of optimal settings for the initial weights, learning rates, and gain exist, where *optimal* can be interpreted in many different meanings: shortest training time, best generalization, *etc.* A proof would follow this sketch: there is without doubt at least one optimal parameter setting. This setting can be used to generate an infinite number of optimal ones by multiplying the learning rate(s) with an arbitrary real value and adapting the other two parameters according to theorem 1 presented chapter 3. The network using these new parameters will give the same result for every input at any state of the training. Obviously, the new setting is also optimal (*q.e.d.*). Note that not necessarily all optimal settings are equivalent in the sense of theorem 1.

Consequently, during the search for an optimal combination of these three parameters, it is sufficient to keep one fixed and to vary only the other two. In contrast to the latter, it is not justifiable on the basis of theorem 1 to hold two parameters fixed and to search only for an optimal value for the remaining one. The experiments will show that this is indeed insufficient.

Several weight initialization methods for multilayer perceptrons have been suggested and are dividable in two groups, those using

1. random or

2. precisely calculated values.

The random weight initialization method is often preferred for its simplicity and ability to produce multiple solutions, as the weights may, due to their initial randomness, converge to various attractors [Kolen-90]. This method was proposed by D. E. Rumelhart *et al.*, who observed that if all weights in a multilayer perceptron are initialized with zero, then they have the tendency to assume identical values during training. Accordingly, they proposed to use random weight initialization in order to avoid this undesired situation by *breaking the symmetry* [Rumelhart-86]. D. R. Hush *et al.* analysed some error surfaces and found "strong support" for initializing the weights to small random values [Hush-92]. From experiments it is clear that the efficiency of this method depends much on the initial weight distribution. Several researchers therefore proposed methods for the determination of suitable weight ranges, respectively variances; an overview is presented in section 4.2.

Methods that calculated precise values for each weight involve extensive statistical and/or geometrical analysis of the data and are therefore very time consuming. The most rigorous among those is the pseudo-inverse method for perceptrons, which, besides being limited to linearly separable data, has several other drawbacks (see [Hertz-91]). Some weight initialization methods are based on special properties of a network, implying that they can not be applied to high order or multilayer perceptrons. An example for such a technique is the weight initialization for radial basis function networks proposed by J. C. Platt [Platt-91].

In order to circumvent the problem of determining an optimal learning rate, an adaptive approach is often used: the learning rate is optimized during the training process. Although several well performing methods exist, a good initial learning rate is still important: both can be expected to speed up the training, independent of whether the other is used or not (see [Moreira-95] or [Schiffmann-92] for overviews on adaptive learning rates). However, M. Moreira shows that the adaptive learning rates can compensate for a bad initial value to a large extend.

4.1 The Choice of the Activation Function

The convergence of the training process, the generalization of the network, *etc.* depends, besides the learning parameters and its topology, also on the sigmoidal function. An easily verifiable fact is that for classification tasks (data with Boolean target values), the linear activation function leads to bad results: the weighted sum of the connection outputs has to be almost exactly 0 and 1 (respectively -1 and 1), which is very restrictive: the training does not only aim at separating the classes of data by adjusting a hyper-surface[1] between them but placing it to a certain distance to all data. Better performance is therefore often obtained if a logistic activation function is used in the output neurons: due the horizontal asymptotes, the network can not "overshoot" the correct output values.

However, high order perceptrons perform better if the hyperbolic tangent is used. The backpropagation favors in this case also weight changes for incorrectly classified patterns with an output **false** which barely takes place if the logistic function is used.

Several researchers use a modified logistic, respectively hyperbolic tangent function in the range $(0, 1.1)$, respectively $(-1.1, 1.1)$ to prevent the weights to move towards $\pm\infty$. In the following, the unscaled function is used, and the Boolean values **true** and **false** are represented by 0.9 and -0.9.

For data sets with continuous valued targets, the choice of the sigmoidal function is more difficult, and arguments similar to those given for the Boolean case are not conclusive. Therefore in the following the experiments for this type of data sets are performed with both the linear and logistic activation function.

4.2 Weights and Learning Rate for Multilayer Perceptrons

In this section a literature overview on weight initialization methods for multilayer perceptrons and the choice of learning rates is given. Some of them, respectively their adaptations to high order perceptrons, will be used latter in this chapter.

S.E. Fahlman performed an early experimental study on the random weight initialization scheme and learning rates for multilayer perceptrons. Based on this study, he proposed to use a uniform distribution in the interval $[-1.0, 1.0]$ but found that the best weight range for the data sets in his study varied from $[-4.0, 4.0]$ to $[-0.5, 0.5]$. Similarly, he observed a wide range of best learning rates [Fahlman-88].

Other researchers tried to determine the optimal weight range using network parameters (see also table 4.1 on page 28):

L. Bottou uses an interval $[-a/\sqrt{d_{in}}, a/\sqrt{d_{in}}]$, where a is chosen in a way that the weight variance corresponds to the points of the maximal curvature of the activation function (which equals 2.38 for the logistic function, 0.66 for the hyperbolic tangent, and is undefined for the identity), and d_{in} is the fan-in (or in-degree) of a neuron. This is done without justifying this interval further in a theoretical manner. L. Bottou trains the neural network

[1]A n-dimensional surface(s) represented by the input vectors for which the output of the network is zero.

only on speech data, without comparing his method to others. The learning rates are fixed and increase with the layer number (from 0.01 to 0.03) [Bottou-88].

J. W. Boers and H. Kuiper initialize weights using a uniform distribution over the interval $[-3/\sqrt{d}_{in}, 3/\sqrt{d}_{in}]$ and a learning rate of 0.4, without any mathematical justification. They state that this interval performed the best on their speech data [Boers-92].

F. J. Śmieja uses uniformly distributed weights which are normalized to the magnitude $2/\sqrt{d_{in}}$ for each node. The thresholds of the hidden units are then initialized to a random value in the interval $[-\sqrt{d_{in}}/2, \sqrt{d_{in}}/2]$, and the thresholds of the output nodes are set to zero. He obtained these values from reasoning about hyperplane spin dynamics and did not validate his method by experiments. An adaptive learning rate is used, but no initial value is given [Śmieja-91] (see appendix B.4 for how to determine an equivalent weight variance).

L. F. A. Wessels and E. Barnard describe two initialization methods. The first method sets the initial weight range to a value which assumes that the output of the network and the target patterns have the same variance. The second method puts equally distributed decision boundaries in the input space (without considering input or output patterns), which produces initial weights for the first layer of connections. The weights of the second layer are set to 1.0. They compared both methods on generalization for three data sets. They found that the second method performed better in terms of generalization, but did not compare convergence speeds or specify a learning rate [Wessels-92]. See appendix B.1 for how the values of their first method are calculated for high order perceptrons.

An approach similar to the first method of L. F. A. Wessels and E. Barnard was introduced by G. P. Drago and S. Ridella. They aim at avoiding flat regions in the error surface by restricting the number of neurons with absolute activations greater than 0.9. They developed a simple formula for the estimation of the best weight initialization scheme for multilayer perceptrons and showed for three data sets that if gives satisfyingly good initial weight ranges. The weights are uniformly distributed over the interval $[-a, a]$, with $a = 1.3/\sqrt{1 + \mathcal{E}[x^2]}$ ($\mathcal{E}[x^2]$ is the expectation for x^2) for the input and $a = 1.3/\sqrt{1 + 0.3 d_{in}}$ for the output layer (assuming that all input values x have the same expected value $\mathcal{E}$). G. P. Drago and S. Ridella did not propose a learning rate [Drago-92]. See appendix B.2 on page 96 for how the formula for the output layer can be translated into an interval, respectively a weight variance.

Y. Lee, S.-H. Oh, and M. W. Kim showed theoretically that the probability of prematurely saturated neurons[2] in multilayer perceptrons increases with the maximal value of weights. They conclude that a smaller initial weight range increases the learning speed of multilayer perceptrons. Simulations performed using two data sets confirm their reasoning, but they disregard that the learning speed of multilayer perceptrons decreases for very small weight ranges. Y. Lee *et al.* are not suggesting an optimal weight range, neither specify a learning rate [Lee-93] (see appendix B.2 on page 96 for an equivalent weight variance).

P. Haffner, *et al.* use a normal initial weight distribution and propose a learning rate $\eta = e^{-4\log(s)+c}$ for a sigmoid of the form $f(x) = s/(1 + e^{-x})$. Unfortunately, they do not compare their approach to others, neither give details (the constant c is not precisely given),

[2]A neuron is called saturated if small weight changes cause only negligible changes of the neuron output

nor justify it mathematically [Haffner-88].

R. L. Watrous and G. M. Kuhn compared a Gaussian to a uniform distribution and found differences on the conditioning of the Jacobian matrix of a neural network but no relation to the convergence speed [Watrous-93].

D. Nguyen and B. Widrow use a multilayer perceptron with piecewise linear activation functions as an approximation of a network having logistic activation functions. Based on this simplification, they calculated an optimal length of $\sqrt[d_{in}]{N_2}$ for the randomly initialized weight vectors and an optimal bias range of $[-\sqrt[d_{in}]{N_2}, \sqrt[d_{in}]{N_2}]$ for neurons in the hidden layer, where N_2 is the number of hidden nodes. The weights the output layer are randomly initialized in the interval $[-0.5, 0.5]$, without any justification given. No learning rate is specified [Nguyen-90].

Y. K. Kim and J. B. Ra calculated a lower bound for the initial length of the weight vector of a neuron to be $\sqrt{\eta/d_{in}}$, where η is the learning rate [Kim-91] (see appendix B.3 on page 97 for an equivalent weight variance).

From these initialization schemes, six can be adapted or applied to high order perceptrons, and are used to calculate the weight variances listed in table 4.1 ($\frac{W}{N_2}$ is the number of connections going to an output neuron). Note that Y. K. Kim gives in contrary to the other only a lower bound for the initial weight variances.

Besides these random initialization methods, some approaches that perform a direct calculation of suitable weights are described here for completeness.

A mixture between a random weight initialization scheme and the pseudo inverse method was developed by C.-L. Chen and R. S. Nutter for perceptrons with one hidden layer. First, the weights in the input layer of the network are initialized with random values. Then, the weights in the second interlayer weight matrix are calculated using the pseudo inverse method applied to the activation values of the hidden layer. C.-L. Chen *et al.* refined this technique further by alternating the adjustment of the first interlayer weight matrix in a backpropagation-like process with the mentioned method of calculating the second interlayer weight matrix. These adjustments are repeated until the network fulfills a certain convergence criterion, and then backpropagation training is started. The authors report faster training in number of iterations [Chen-91], but they disregard the computational complexity of the matrix inversions.

T. Denoeux and R. Lengellé initialize a one hidden layer perceptron with prototypes. This method requires a transformation of the input patterns to vectors of unit length and increased size. Additionally, prototypes have to be found by a cluster analysis. The authors reported improvements in training time, robustness versus local minima, and better generalization. The learning is adapted after each training step [Denoeux-93].

It should be remarked that the various methods for a dynamic adaption of the learning rate are neglected on purpose, as they are supposed to decrease the training time independent of the initial learning rate (compare [Moreira-95] and [Schiffmann-92]). This means that they always can used as "add-on". On the other hand, these methods need more time find a good learning rate if it is initially very badly chosen. Additionally, these methods

	Ω	$\frac{W}{N_2}$	Bottou	Boers	Wessel	Drago	Kim	Śmieja
Linear activation function								
CES	2	6	-	0.5	0.4	0.2	0.08η	-
	3	10	-	0.3		0.1	0.03η	-
Auto-mpg	1	8	-	0.4	0.5	0.2	0.05η	-
	2	36	-	0.8	0.03	0.05	0.002η	-
Solar	1	13	-	0.2	0.3	0.1	0.02η	-
	2	79	-	0.04	0.007	0.02	0.0005η	-
Servo	1	13	-	0.2	0.3	0.1	0.02η	-
	2	79	-	0.04	0.007	0.02	0.0005η	-
Glass	1	16	-	0.2	0.3	0.1	0.01η	-
	2	121	-	0.03	0.003	0.02	0.0002η	-
Logistic activation function								
CES	2	6	0.3	0.5	1.7	0.2	0.08η	0.1
	3	10	0.2	0.3		0.1	0.03η	0.04
Auto-mpg	1	8	0.2	0.4	2.0	0.2	0.05η	0.06
	2	36	0.05	0.8	0.1	0.05	0.002η	0.003
Solar	1	13	0.2	0.2	1.2	0.1	0.02η	0.02
	2	79	0.02	0.04	0.03	0.02	0.0005η	0.0006
Servo	1	13	0.2	0.2	1.2	0.1	0.02η	0.02
	2	79	0.02	0.04	0.03	0.02	0.0005η	0.0006
Glass	1	16	0.1	0.2	1.0	0.1	0.01η	0.02
	2	121	0.02	0.03	0.01	0.02	0.0002η	0.0003
Hyperbolic tangent activation function								
Br. vowels	2	66	0.002	0.05	0.009	0.03	0.0002η	0.001
Wine	2	92	0.002	0.03	0.005	0.02	0.0001η	0.005
Monk 1-3	2	154	0.0009	0.02	0.002	0.01	$4*10^{-5}\eta$	0.0002
Fi. vowels	2	231	0.0006	0.01	0.0009	0.008	$2*10^{-5}\eta$	$8*10^{-5}$
Digits	2	2081	0.0009	0.001	$1*10^{-5}$	0.0009	$7*10^{-7}\eta$	$9*10^{-7}$

An entry '-' means that this method could not be applied.

Table 4.1: Initial weight variances as calculated by different authors.

often introduce one or more additional parameters, which reduces the user-friendliness. Another drawback is that most of these methods require off-line learning, which is commonly supposed, although not proven, to be slower than on-line training (see [Hertz-91] on page 119).

4.3 Random Weight Initialization

The examination of the weight initialization methods enumerated in section 4.2 shows that researchers try to optimize learning speed and generalization performance of neural networks initialized with random weights mainly in two ways:

1. by using different distribution shapes for the weights or

2. estimating good initial weight variances[3] for example from:
    - the steepness of the sigmoidal function (indirectly determined by the maximal points of curvature by L. Bottou),
    - the number of connections feeding into a neuron,
    - constants motivated by experiments,
    - analysis of the data, and
    - the number of connections.

As one expects both the distribution shape and variance to have an almost independent influence on the training, they are examined in separate experiments.

4.4 Best Weight Distribution Shape

Three different initial weight distribution shapes are compared for an influence on the convergence speed of high order perceptrons:

- uniform,
- normal or Gaussian[4], and
- a novel distribution which is uniform over the intervals $[-2a, -a]$ and $[a, 2a]$ and zero everywhere else[5].

It is unknown, which attribute[6] makes weight distributions comparable in terms of convergence speed of neural networks or their generalization. Therefore, for all distributions the variance is varied over a wide range, and for each distribution shape the best result is used to compare it with the others.

For the simulations performed (the outcome of these experiments is documented in appendix C on page 99), a suboptimal learning rate is used[7], as it is very laborious and computing time consumptive to find the optimal learning rate for each combination of data set and network. The only optimization technique applied to speed up learning is flat spot elimination [Fahlman-88].

The tables C.1, C.2, and C.3 show that the shortest training time per data set does not depend on the shape of the initial weight distribution (the differences are in the range of the statistical error and therefore insignificant).

[3]For a uniform distribution over the interval $[-u, u]$ the variance σ^2 equals $\frac{u^2}{3}$.

[4]Neural network weights are often assumed to be normally distributed [Bellido-93].

[5]This distribution was included into this research when the difference between the other two was found to be minimal.

[6]This attribute can be the variance, the range, or some other value.

[7]The experiments are preceeded by some simulations in which it was tried to find suitable values in order to keep the computational effort reasonable.

The main difference between the outcome of the experiments is the location of the optimal initial weight range: for the normal distribution this range can on average include slightly higher values as compared to the other two distributions. This shows that it is impossible to make a fair comparison of initial weight distributions on the base of one specific variance or range.

It can be expected that multilayer perceptrons behave similar, which might explain the better performance of the normal distribution reported by P. Haffner *et al.*

4.5 Weight Variance and Learning Rate

In order to set in relation the initial weight variance, learning rate, training time, and the generalization, as well as to evaluate the weight initialization techniques and the determination of the best learning rate methods, the following is done:

1. The optimal learning rate and weight initialization variance for fast convergence are globally searched for several data sets and three different sigmoidal functions of a fixed gain. These functions are the hyperbolic tangent, the logistic, and the identity (linear) function. A search for an optimal gain is not required, as any network can be "normalized" to have only activation functions with a gain equal to 1 (compare theorem 1).

2. Similarly, the optimal learning rate and weight initialization variance for good generalization are searched for.

3. The outcome of these experiments is used to estimate the efficiency of the heuristics for the estimation of the optimal weight range.

The search for an optimal combination of learning rate and weight initialization variance can be done theoretically by a *line-search* algorithm, assuming that the average training time, respectively the generalization, forms a function with smooth surface: the gain is kept to a standard value of one, and either the weight initial range or the learning rate is varied until an optimum for both values is found. Unfortunately, the results of a simulation are subject to statistical fluctuation. Neural networks will usually not converge in the same number of cycles and have an equal generalization for two different sets of initial weights, even if they are in the same distribution. Consequently, first a line-search algorithm is used to get close to the optimum. Then, its surrounding is searched by performing experiments for each combination of learning rate and weight variance.

During the experiments described in the rest of this chapter, the networks are considered to have converged if the error for the training set was smaller than in table 4.2 on the next page.

Data set	precision on training set
Solar	MSE smaller than 0.06
CES	MSE smaller than 0.1
Monk 1-3	100% correctly classified
Auto-mpg	MSE smaller than 0.06
Glass	MSE smaller than 0.03
Servo	MSE smaller than 0.07
Wine	100% correctly classified
Digits	99% correctly classified

Table 4.2: The convergence conditions for the experiments concerning the optimal choice of training parameters.

4.6 First Results

To give an overview on the behavior of the required training time as a function of weight variance and learning rate a series of experiments using the solar data set is discussed in more detail. The outcome of these experiments is shown in figure 4.1, where the training time is displayed as a function of learning rate and initial weight variance. The contour plot beneath the graph demonstrates its channel-like shape with an outlet where the weight variance is zero. It can be seen that the convergence time stays almost constant for weight variances in the interval $[0.0, 0.1]$ and the same learning rate. If the latter is well-chosen, and the weight variance is optimal, then the high order perceptrons always converge in a near-optimal number of training cycles. The shape of the plot in figure 4.1 is common to all the experiments performed during this study, only the location and the narrowness changed with the data set, the order of the network, as well as other parameters.

Interestingly, the optimal learning rate for high order perceptrons is sometimes, as for this example, well above 1.0. In a stark contrast to this observation is the common belief that the learning rate has to be below 1.0 for a "standard" setting[8] of the other training parameters. However, a general rule for the determination of the learning rate can not be given.

The behavior of multilayer perceptrons is different: in figure 4.2 on page 33 it can be seen that the training time as a function of the weight variance and learning rate has a bowl-like shape (the multilayer perceptron is trained on the solar data set). The rough shape of this graph is probably representative for most multilayer perceptrons and data sets as it was observed during several experiments (only the location of the minimum and the narrowness changed). Also, multilayer perceptrons usually fail to converge for weight variances equal to zero, and their training is slow if the initial weights are very small (as already stated S. E. Fahlmann). In figure 4.2 it is shown that the optimal learning rate multilayer perceptrons can be bigger than one.

[8] A "standard" setting would include a steepness of 1.0. Any other learning rate could be used if the steepness is not defined.

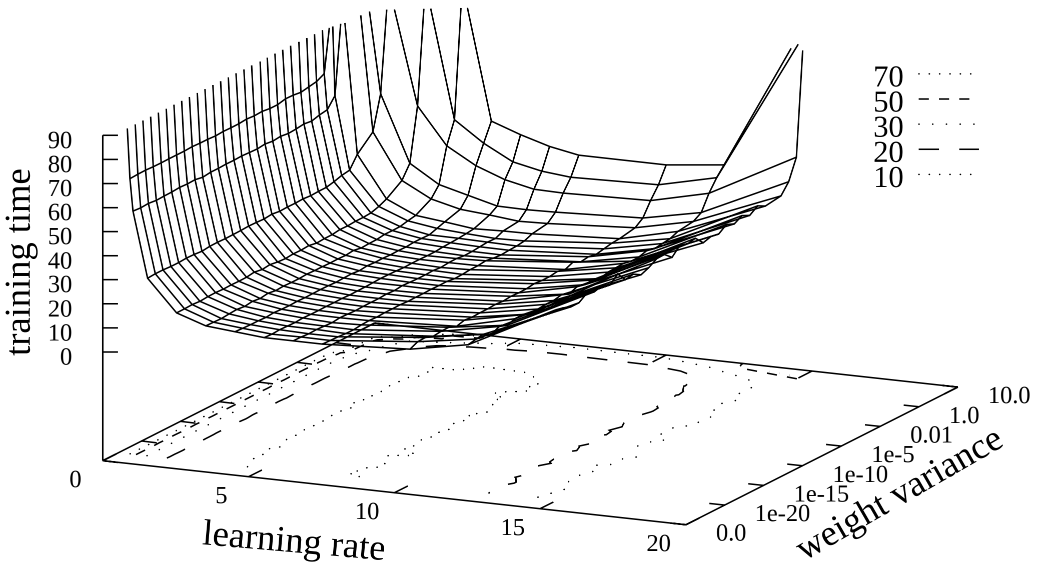


Figure 4.1: The training time of a high order perceptron as a function of weight variance and learning rate for the solar data set.

The average generalization performance of high order perceptrons, that have the same topology and are trained on the same data set, as a function of the learning rate and initial weight variance is displayed in figure 4.3. It can be seen that, similar to the training time, the generalization error is almost constant if the initial weight variance is below a certain value and the learning rate is unchanged. Furthermore, the generalization error increases for values above this limit (only a few exception to this behavior were encountered among 27 series of experiments). For a constant weight variance below this limit, the generalization improves (the error decreases) with a decreasing learning rate - just up to the point where the high order perceptrons cease to learn (do not converge in a certain number of iterations). This point is symbolized by the gray bar in figure 4.3.

Multilayer perceptrons behave similarly, as it is shown in figure 4.4 (experiments have been performed for the solar, the wine, the glass and the servo data set). The most important difference to high order perceptrons is, as the networks do not or only very slowly converge for a weight variances close to zero, such variances should not be used.

However, as figure 4.3 displays the average over many simulations is somewhat misleading: the minimal error observed for all pairs of learning rate and initial weight variance for which the high order perceptrons converge is almost constant. Only the upper limit of the interval, in which errors can be observed, depends on the learning rate and weight variance. In other words, the minimal error is constant, whereas its maximum varies, as shown in figure 4.5, where the lower and the upper graph are the minimal, respectively the maximal, error as a function of learning rate and weight variance.

In contrast to the behavior of the learning time as described above, the generalization performance decreases for some data sets before the network ceases to learn due to a too

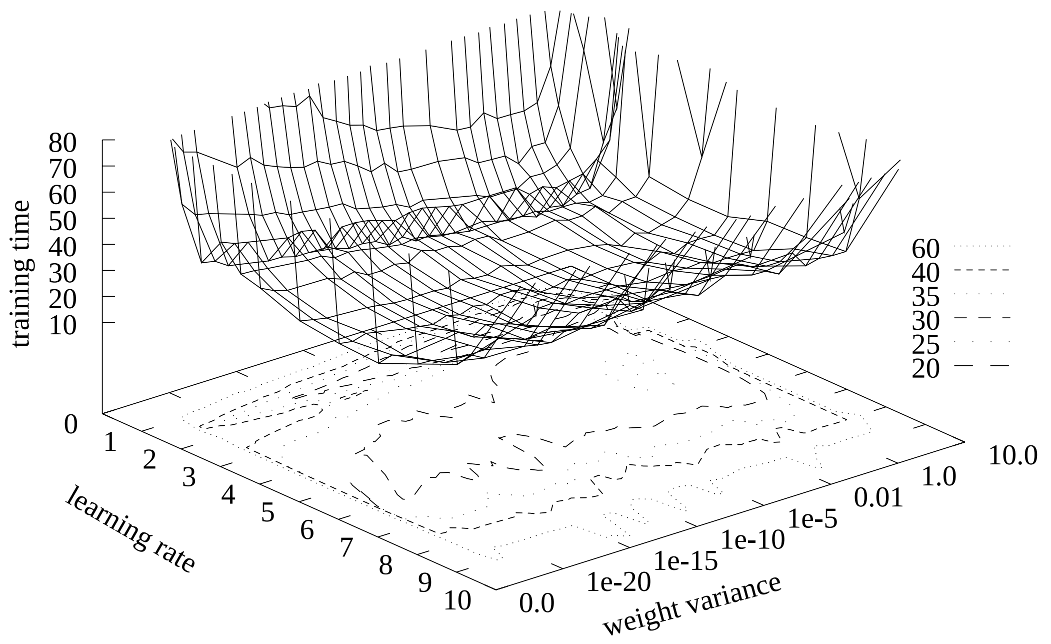


Figure 4.2: The training time of a multilayer perceptron as a function of weight variance and learning rate for the solar data set.

small learning rate (giving a similar graph as for the learning rate in figure 4.1). Another behavior shows a network with a logistic activation function trained on the CES data set: the generalization decreases together with a decreasing learning rate. More precisely, the distance between the minimal and maximal generalization, as displayed in figure 4.6, is almost constant over the whole range of learning rates and weight variances.

4.7 Evaluation of the Experiments for Fast Learning

Table 4.3 shows a resume of the approximately 800,000 simulations performed. It lists the combinations of initial weight variances and learning rates for which the convergence time is (almost) optimal for the different activation functions and network orders. In the right most column, the 95% confidence interval for the on average required number of training cycles is given.

Globally, for fixed learning rate not far from the optimum, a weight variance exists below which the network converges in almost the same number of iterations. This includes zero weights. Above a certain weight variance, the convergence time increases very fast.

Comparing the estimated optimal weight variances in table 4.1 on page 28 with these results, these methods give rarely a good estimation of the upper limit for the weight variance. However, as the network converges in an almost optimal time if all weights are zero or very close to, there is no reason to use a higher value.

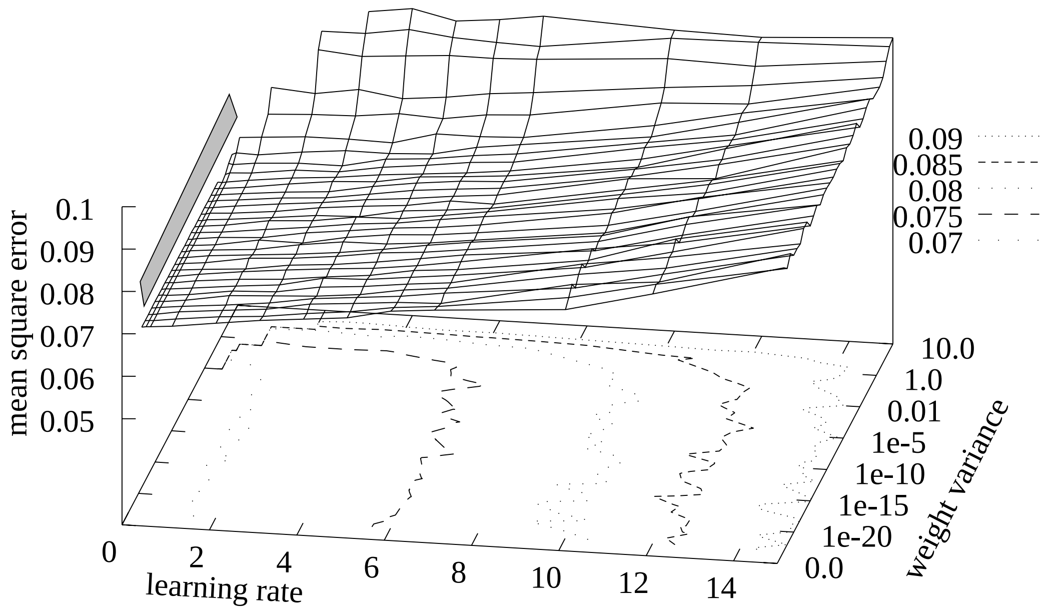


Figure 4.3: Generalization of a high order perceptron as a function of weight variance and learning rate for the solar data set.

It can be easily seen that the activation function has an important influence on the optimal learning rate: it is for high order perceptrons with a logistic activaton function on average 25 times higher than for the linear activation function. This factor varies between 5 and 85 for the different data sets. This behavior is at least partly related to the lower value of the first derivative of the logistic as compared to the linear activation function ($\frac{1}{4}$ at zero and even smaller for other values). Supposed that the logistic activation function is scaled to have a first derivative of one at zero, then the learning rate has to be divided by 16 in order to obtain the same network behavior. This number compares well with the difference between the optimal learning rate for the linear and logistic activation function.

For the linear activation function, optimal learning rates between 0.02 and 0.6 have been observed. Surprisingly, for the logistic activation function this range is $[0.5, 7.0]$, where most rates are above 1.0, although usually a learning rate smaller than 1.0 is recommended. The range $[0.005, 2.5]$, in which optimal learning rates have been found for the networks with the hyperbolic tangent activation function and trained on the classification problems, is very big as compared to the approximation problems. However, it is unlikely that the activation function causes this behavior but the data sets and the target patterns being Boolean (compare theorem 1).

The choice of the activation function changes also the convergence time: networks with the logistic activation function converge on average faster than those with a linear one. For the solar data set, the network using logistic functions is not able to attain the same precision as a network with a linear activation function. *Vice versa*, first order perceptrons with a logistic activation function learn the servo data set up to a higher precision than the

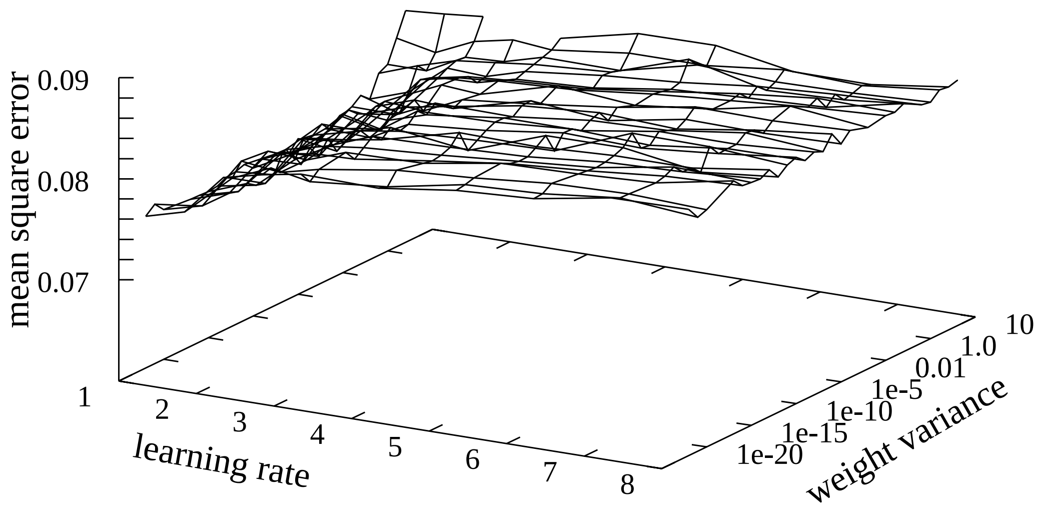


Figure 4.4: Generalization of a multilayer perceptron as a function of weight variance and learning rate for the solar data set.

ones with the linear activation function.

The estimation of the optimal learning rate according to P. Haffner *et al.* does not depend on the number of inputs or connections. This coincides with table 4.3 which reveals no correlation of these parameters and the optimal learning rate. On the contrary, the experiments performed with the solar and servo data sets show that two networks can have the same topology but different optimal settings for the learning rate and weight variance. These special settings for these parameters can therefore be regarded as a property depending on the information contents of the data set. However, those values are only of little help: the range of near-optimal learning rates is high, and a learning rate can cause non-convergence for a certain data set, although it is optimal for another. This is especially observable for the classification problems. No value for the constant c in his formula exists which is effective for all data sets.

The method of Y. K. Kim *et al.* does not match the outcome of the experiments performed with the high order perceptrons, as he excludes very small weight variances.

The optimal settings seem also independent of the complexity of the problem: the servo data set, which is supposed to be difficult to learn (which is confirmed as the high order perceptrons need a higher number of training cycles to learn this data set as compared to others), has an optimal learning rate comparable to "simpler" data sets (as for example the solar data set). Furthermore, no working theory like **complex data $\rightarrow$ low learning rate** or **complex data $\rightarrow$ high learning rate** can be confirmed by the experiments. This applies equally to hypotheses based on the number of inputs, outputs, or patterns.

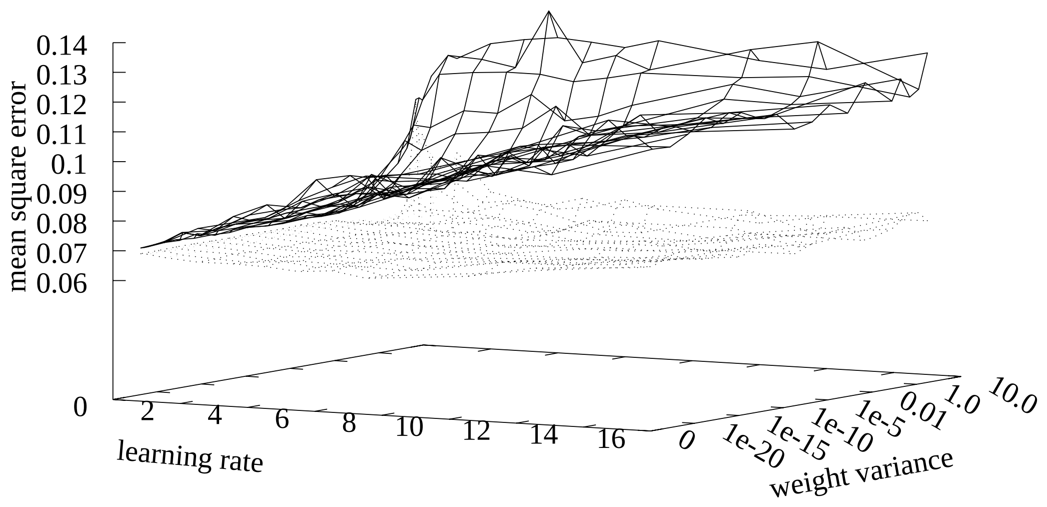


Figure 4.5: Minimal and maximal error as a function of weight variance and learning rate for the solar data set.

4.8 Evaluation of the Experiments for Good Generalization

Table 4.4 shows the ranges of initial weight variances and learning rates for which the high order perceptrons performed best on the test data in terms of generalization.

In all but one experiment, high order perceptrons initialized with zero weights, or random values of a variance close to zero, performed optimally. The exception is represented by a first order perceptron with a linear activation function trained on the servo data, which has a better generalization for initial weight variances above 3.0 (experiments were performed for variances up to 10^5, for which the training time was about 11 times as high as for a zero weights and 5 times as compared to a weight variance of 3.0).

No activation function is preferred in general since for three experiments the networks with the logistic function yield a better performance and during three experiments those where linear activation functions are used (in the other cases the differences are within statistical error margins, or the results are not comparable due to different training error).

Note that the generalization of the networks with all initial weights equal to zero does not end up in the same solution for each simulation. The presentation of the patterns in a random sequence is sufficient to diversify them. Furthermore, the variance of the generalization on trained networks initialized with zero weights is usually similar to those for initial weight variances in the range of optimal values. This leads to the conclusion that the diversity of the solutions for zero and small random weights is equal. However, the latter may perform better for data sets for which a random presentation of the elements is insufficient to prevent that the weight assume similar values[9], although this behavior was

[9]Compare [Rumelhart-86].

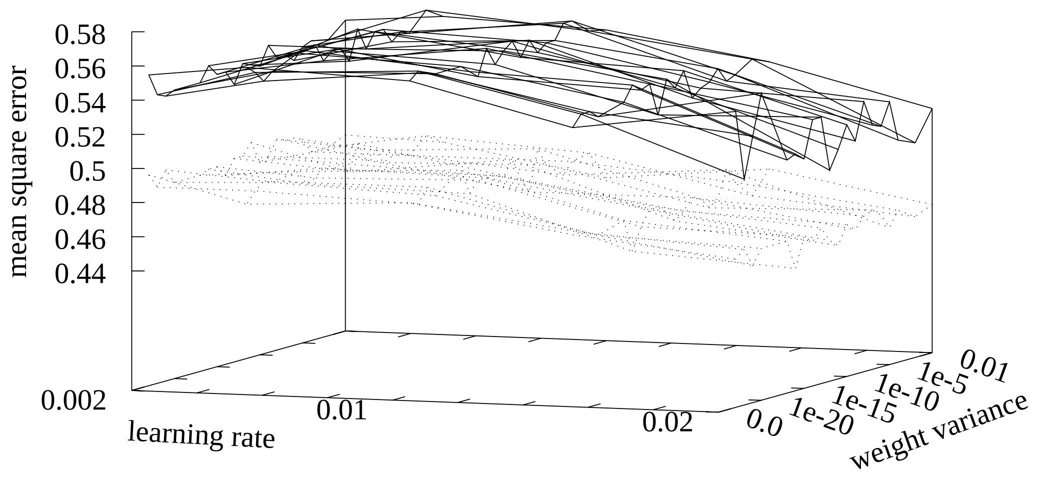


Figure 4.6: Minimal and maximal error as a function of weight variance and learning rate for the CES data.

not observed during all the experiments with high order perceptrons.

The optimal learning rate for the different data sets seems to follow no rule: a correlation between the number of connections and the optimal learning rate is not visible, even if only networks of the same order are considered.

Comparing the optimal learning rates for fast convergence with those for a good generalization, the following can be observed: for the linear activation function and the hyperbolic tangents, the values are equal or similar. This behavior differs for the logistic activation function: the learning rate for fastest convergence is almost always higher as compared to those for best generalization.

4.9 Conclusion

For all the data sets, I found a certain upper limit for the initial weight variance, below which both the network convergence and generalization are near-optimal (only one exception was observed for 27 series of experiments). In contrast to the multilayer perceptrons, even an initialization with zero weights gives optimal results if the learning rate is well-chosen. Consequently, a near-optimal generalization can be achieved with an initialization of high order perceptrons using zero or very small random weights. The latter choice should be preferred in order to prevent trouble with exceptional data sets. However, the use of all initial weights equal to zero does not prevent the networks to assume different solutions: the presentation of the patterns in a random order (which is in any case advantageous to ensure and accelerate convergence) is sufficient to randomize the weights.

The optimal initial weight variance depends on the data set for both an optimal training time or generalization. These values do not depend in a visible way on the number of connections or the order of the network. None of the weight initialization methods originally proposed for multilayer perceptrons is able to predict this value.

A method for the determination of an optimal learning rate could not be found. Even worse, my experiments show that a method using only parameters of the network topology, such as the number of connections, the type and steepness of the activation function, or the order of the network is most likely to fail. The optimal learning rate probably depends mainly on the clustering of the data and is therefore impossible to estimate in a simple way. However, the shape of the activation function changes the range of optimal learning rates (see the tables 4.3 on the facing page and 4.4 on page 40 for these ranges) which further depends on whether one optimizes training time or generalization. The best generalization can even be observed for learning rates which sometimes cause slow or non-convergence.

Linear activation function				
	Learning rate	Weight variance	Ω	Iterations
CES	0.5 - 0.6	0.0 - 0.1	2	14.9±4
CES	0.4	0.0 - 0.1	3	11.7±5
Auto-mpg[A]	0.1 - 0.15	0.0 - 0.01	1	10.4±6
Auto-mpg	0.1	0.0 - 0.01	2	34.3±33
Solar	0.2 - 0.3	0.0 - 0.001	1	5.3±4
Solar	0.2	0.0 - 0.0005	2	4.1±2
Servo[B]	0.07	0.0 - 0.001	1	3.0±3
Servo	0.12	0.0 - 0.1	2	166±4
Glass	0.1	0.0 - 0.01	1	8.7±5
Glass	0.02	0.0 - 0.0001	2	6.0±2
range	[0.02, 0.6]	max: [0.0001, 0.1]		
Logistic activation function				
CES	3.0	0.0 - 0.5	2	15.4±6
CES	2.0	0.0 - 0.01	3	10.1±4
Auto-mpg[A]	2.0 - 4.0	0.0 - 0.01	1	1.6±1
Auto-mpg	1.5 - 2.0	0.0 - 0.2	2	34.0±35
Solar[A]	5.0 -7.0	0.0 - 0.01	1	1.7±1
Solar	5.0 - 7.0	0.0 - 0.1	2	8.7±4
Servo	6.0 - 7.0	0.0 - 1.0	1	31.8±12
Servo	4.0 - 5.0	0.0 - 0.2	2	15.8±3
Glass	2.0	0.0 - 0.2	1	8.1±4
Glass	0.5	0.0 - 0.01	2	5.3±4
range	[0.5, 7.0]	max: [0.01, 0.5]		
Hyperbolic tangent activation function				
British vowels	0.005	0.0 - 0.0001	2	55.0±10
Wine	2.5	0.0 - 0.2	2	36.4±36
Monk 1	0.05 - 0.07	0.0 - 0.01	2	3.4±1
Monk 2	0.05	0.0 - 0.01	2	50.4±11
Monk 3	0.05	0.0 - 0.005	2	14.8±9
Finish vowels	1.5 - 2.0	0.0 - 0.2	2	54.3±14
Digits	0.1	0.0 - 0.01	2	13.3±7
range	[0.005, 2.5]	max: [0.0001, 0.2]		

[A]The error of 0.06 could not be reached, 0.075 is used instead

[B]The error of 0.07 could not be reached, 0.13 is used instead

Table 4.3: Best settings for learning rate and weight variance for high order perceptrons with a gain of 1 if fast learning is important.

Linear activation function					
	Learning rate	Weight variance	Ω	W	error
CES	0.5 - 0.7	0.0 - 0.1	2	2	0.073±1
CES	0.6	0.0 - 0.5	3	6	0.070±2
Auto-mpg^A	0.1^C	0.0 - 0.1	1	8	0.066±1
Auto-mpg	0.15	0.0 - 0.005	2	36	0.057±1
Solar	0.1	0.0 - 0.1	1	13	0.073±1
Solar	0.004 - 0.005	0.0 - 0.0001	2	79	0.072±0
Servo^B	0.2	>3.0	1	13	0.126±1
Servo	0.12 - 0.15	0.0 - 0.1	2	79	0.112±1
Glass	0.15	0.0 - 1.0	1	16	0.027±1
Glass	0.04	0.0 - 0.001	2	121	0.035±1
range	[0.004, 0.7]	max: [0.0001, 1.0]			
Logistic activation function					
CES	1.0 - 2.0	0.0 - 0.5	2	2	0.082±1
CES	1.0	0.0 - 0.1	3	6	0.086±1
Auto-mpg^A	0.2 - 2.0	0.0 - 0.01	1	8	0.063±1
Auto-mpg	0.3 - 0.4	0.0 - 0.2	2	36	0.057±1
Solar^A	3.0 - 20.0^C	0.0 - 5.0	1	13	0.084±1
Solar	0.3 - 1.0^C	0.0 - 0.1	2	79	0.069±1
Servo	1.0^C	0.0 - 2.0	1	13	0.080±0
Servo	0.1 - 7.0	0.0 -0.1	2	79	0.0118±1
Glass	3.5^C	0.0 - 2.0	1	16	0.026±1
Glass	0.9	0.0 - 0.01	2	121	0.034±1
range	[0.1, 20.0]	max: [0.01, 5.0]			
Hyperbolic tangent activation function					
Br. vowels	0.005 - 0.01	0.0 - 0.01	2	726	47.8±2%
Wine	2.5^C	0.0 - 0.7	2	276	12.1±5%
Monk 1	0.1^C	0.0 - 0.05	2	154	10.9±5%
Monk 2	0.05^C	0.0 - 0.05	2	154	17.2±5%
Monk 3	0.07^C	0.0 - 0.01	2	154	16.2±3%
Fi. vowels	2.0^C	0.0 - 0.2	2	1,155	20.9±7%
Digits	0.01 - 0.02^C	0.0 - 0.01	2	20,810	4.0±1%
range	[0.005, 2.5]	max: [0.01, 0.7]			

^{A}A max. error of 0.06 could not be reached, 0.075 is used instead

^{B}A max. error of 0.07 could not be reached, 0.13 is used instead

^{C}Better gener. is observable for learn. rates causing sometimes non-convergence.

The error is given as mean square diff., resp. as percent misclassification (%).

Table 4.4: Best settings for learning rate and weight variance for high order perceptrons with a gain of 1 if good generalization is important.

Chapter 5

Pruning of Neural Networks

5.1 Introduction

One of the most important problems encountered in the practical application of neural networks is to find a suitable or, ideally, minimal neural network topology. One reason is that an unsuitable topology increases the training time or even cause non-convergence, and it is likely to decrease the generalization capability of a network [Gosh-94]. Additionally, there are economical and technical arguments to prefer small networks: the price for hardware implementations is directly related to the surface or number of chips and therefore the network size. Similarly, software implementations for oversized neural networks are far less efficient.

A basic approach to find nearly minimal topologies are, apart from constructive and growing approaches (see chapter 6 and 7), *pruning* methods. Generally, the latter method starts the training with a neural network which is expected to be big enough to ensure a successful training. Then, when the neural network is estimated being too big or simply the network training succeeds, some connections or neurons are removed and the training resumed, until a suitable topology is found.

Independent of the pruning method, the evaluation of its quality is difficult to perform, mainly because of a lack of neural network *optimality criteria*. A primary criterion, the *minimal network topology*, can be defined but is very difficult to determine for a specific application [Fiesler-93]. A proof for a certain neural network topology being actually minimal, appears to be almost impossible, and with the current knowledge the required effort is beyond the scope in industrial development. Besides this, the minimality criteria varies for different implementations and applications. Such a criteria can be the number of layers, neurons, and connections, or even a mixture of these. Furthermore, various implementation dependent constraints may play a role in the choice of topology: for some types of hardware implementations not only the total number of connections and neurons is important but also, for example, the maximal number of connections going to and from a neuron (these numbers are often called the *fan-in* and the *fan-out* of a neuron). Or, if the application has tight real time requirements, the depth of a multilayer perceptron may, for example, not exceed one hidden layer.

The other important criterion, *generalization*, is, besides being improperly defined, also application dependent. In general, tradeoffs between training time, network size, and generalization should be taken into account when comparing pruning methods, which adds another dimension to the problem of defining an optimal neural network.

The numerous pruning (and constructive) algorithms and the fact that their efficiency is usually uncompared can be regarded as proof for the encountered difficulties. Summaries of those pruning methods and heuristics, without further comparison or experimental results, are for example given in [Reed-93] and [Wynne-Jones-91]. It seems to be impossible to compare methods for the optimization of neural networks in a theoretical way. The comparison of these methods is therefore performed on the base of experiments. This requires optimality criteria to be established, and a framework in which the pruning methods are applied to neural networks. This framework should examine factors like the generalization of the network, the total training time, the complexity of the pruning method, and the implementation costs. Furthermore, results need to be supported by the analysis of a large number of experiments, which guarantees a high statistical confidence level (compare appendix A). This, however, is often neglected [Prechelt-94]. A framework, that can be used for the comparison of pruning methods, is presented in the following and applied to high order perceptrons. This or a similar framework is applicable to other types of neural networks and constructive methods. It can be expected that some observations and results presented in the following are also valid for other feedforward neural networks.

5.2 Connection Pruning Methods

A pruning algorithm basically has to decide *which unit(s)* to prune, *when to prune*, and *when to stop* the training. The choice of the pruning method, which selects the units to be removed, is the most crucial part of a pruning algorithm. Therefore the other two concerns are unlikely influenced by the choice of the pruning method, optimizations to them are not considered in this research. Due to practical constraints, a simple strategy is used instead: a network is pruned by one or a few connections, whenever its training converged. Then, the training is eventually aborted when the network ceases to learn. This approach has an advantage: as the networks are pruned until they cease to re-learn the training data, it is possible to verify whether some of the assumptions made for other types of neural networks also apply to high order perceptrons (see section 5.4).

For comparison, five weight removal heuristics were chosen based on their low computational complexity and applicability to high order perceptrons. These methods emerge from the idea to minimize the error induced by the removal of the unit or, in other words, to estimate the *sensitivity* of the neural network to the removal of a certain connection.

1. The simplest heuristic selects the connections with the smallest weights. In addition to the plain method, which removes only connections, the growth of the error is reduced

by adding the connection's *mean contribution* $\frac{1}{P}\sum_{p=1}^{P} c^p$ (its mean value averaged over the training set) to the bias of the neuron receiving input from the removed connection [Sietsma-91].

OBD, as well as OBS (Optimal Brain Surgeon, see [Le Cun-90]) have be extended to γOBD and γOBS by M. W. Pedersen *et al.* These methods calculate the sensitivity with respect to the error and a regularization term.

2. J. Sietsma and R. J. F. Dow proposed to remove the connections with the smallest contribution variance $\sigma_p(c^p)$. The method is therefore called the smallest variance ($min(\sigma)$) method. The mean output of the removed connection is added to the corresponding bias [Sietsma-91].

3. E. D. Karnin estimates the sensitivity s of a connection c by:

$$s = \sum_{n=1}^{N} (\Delta w(n))^2 \frac{w^f}{\eta(w^f - w^i)},$$

   where w^f is the weight in the current training epoch n, w^i the initial weight, and $\Delta w(n)$ the weight change in the n-th epoch [Karnin-90]. This formula has a problem: there is no theoretical reason for the denominator always being non-zero. Experiments show that this happens sometimes. As E. D. Karnin neglects this case in his publication, it is decided here to set in this case the whole fraction to zero. s is therefore calculated using:

$$s = \sum_{n=1}^{N} \begin{cases} (\Delta w(n))^2 \frac{w^f}{\eta(w^f - w^i)} & \text{if } w^f \neq w^i \\ 0 & \text{else.} \end{cases}$$

4. M. C. Mozer and P. Smolensky developed a weight removal method called *skeletonization* which estimates the error induced by the removal of a unit by multiplying its output with an *additional strength.* Then, with having in mind that setting an additional strength of zero is equivalent to removing it, those units are removed for which the derivative of the error function to these additional strengths are small (actually, they use an exponentially decayed average of these values) [Mozer-89].

5. W. Finnoff *et al.* define a test statistic for the probability that a weight becomes zero. Using this probability, a connection is removed if the probability that it becomes zero is high. This sensitivity measure is integrated into a pruning method called *autoprune* [Finnoff-93]. This pruning method is extended to the pruning method $\lambda-$*prune* by L. Prechelt in order to determine how many units should be pruned at each step [Prechelt-95]. However, in the experiments described in the following, only the test statistic is used as sensitivity measure.

There is another reason for using only the formula for the calculation for the sensitivity of the neural network for the removal of a connection and not the whole framework of a pruning method: it is rather difficult to find good settings for the various additional parameters involved in the algorithms which decides upon when to prune and when to stop the training. This problem became apparent during a project on pruning multilayer perceptrons [Beuchat-95]. These parameters also strongly affect the user-friendliness and inhibit a fair comparison with the embedded sensitivity estimation. Note that the framework of E. D. Karnin, which J.-L. Beuchat found to be best for multilayer perceptrons, is incompatible with the theoretical justification of OBD as the network is pruned before it converges: OBD assumes that $\Delta\mathbf{w} = 0$, or, equivalent, the network has converged into a (local) minimum.

Other pruning methods were excluded from this study because they were unsuitable for high order perceptrons or have a high computational complexity. The latter class includes methods, which use a full Hessian matrix, like for example flat minimum search [Hochreiter-96][1] and Optimal Brain Surgeon (OBS) [Hassibi-92].

Weight decay was excluded, as it was found to be not effective by S. J. Hanson *et al.* [Hanson-89]. Their publication throws also a negative light on methods using penalty terms[2]. They observed that about 75% of the simulations where weight decay was used, failed to converge, whereas the backpropagation without a penalty term always converged under the same conditions. Other, similar methods are summarized by E. D. Karnin [Karnin-90] and G. Fahner [Fahner-92]. Why methods using penalty terms may fail to converge is exemplarily shown for the penalty term proposed by A. S. Weigend *et al.* [Weigend-90]:

$$\lambda \sum_i \frac{\frac{w_i^2}{w_0^2}}{1 + \frac{w_i^2}{w_0^2}}$$

Let the value of w_0 in this penalty term be 1 and the weight of the penalty term λ in the objective function be 0.4. Then the penalty term has its minimum at $\mathbf{w} = 0$ and approaches $\lambda = 0.4$ for large weight vectors (compare the dashed curve in figure 5.1). It is obvious that if this term is included into the objective function, then minima of the error term, which are equivalent to a weight vector close to $w = 0$, are preferred. Now, suppose that the error surface is shaped as the solid curve in figure 5.1 with a global minimum for $w \approx 2$. The addition of these two curves illustrates the impact on the (fictive) error surface represented as the dotted curve: the former harmless local minimum at $w \approx 0.3$ replaces the former global minimum, the network will therefore very often fail to learn properly.

5.3 Experiments

The experiments described in the following consist of at least 100 simulations each; if the outcomes of two experiments are compared, the confidence in difference was calculated and ensured to be at least 99%.

Each simulation performed follows the same scheme:

[1] Also published as technical report [Hochreiter-94]

[2] Penalty methods are based on modifications of the objective function designed to penalize certain weights.

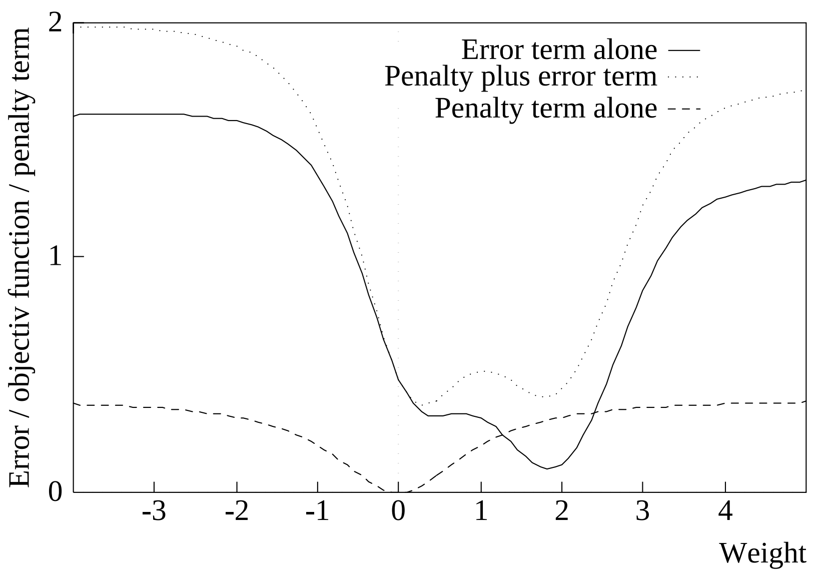


Figure 5.1: The penalty term may change the location of global minima.

Data set	Precision on training set
Solar	MSE smaller than 0.05
CES	MSE smaller than 0.08
Monk 1-3	100% correctly classified
Auto-mpg	MSE smaller than 0.06
Glass	MSE smaller than 0.03
Servo	MSE smaller than 0.05
Wine	100% correctly classified
Digits	99% correctly classified

Table 5.1: The required precision on the training sets for the pruning experiments.

1. Initialization of the network of order n with random weights (compare chapter 4). During a real application of high order perceptrons, the order n would be initially chosen to be one and then increased by one whenever it is clear that the corresponding network is unable to learn the task. However, for the research purposes, various orders are examined (see for example table 5.2 on page 49).

2. Application of the backpropagation algorithm until the in table 5.1 given percentage of training data are correctly classified by the network, respectively the mean square error on the training set reaches the value listed in this table (see table 5.2 for the learning rate).

3. Removal of connections from the network. Depending on the size of the network, from one connection up to 5% of the connections are removed from the network (see

remark 2 below). The connection(s) to be removed are those found to have the smallest sensitivity calculated according to the methods listed in section 5.2.

4. If required, the network is retrained until it reached the performance on the training data as listed in table 5.1, or a certain limit of training steps has passed[3]. If the network converged, pruning is resumed (step 3). If not, the last net found is accepted as a solution.

Remark 1: for data sets with a high number of elements equal zero per input vector, as for example the digits data set, many second order connections have a constant value of zero for all inputs and can therefore removed without lost. However, this was not done in these experiments in order to test the pruning methods.

Remark 2: if the pruned network is very big compared to the minimal network, the pruning of a connection often did not require any retraining. During the simulation described in the remainder of this chapter, the removal of the first connections the error remains usually unchanged and below the value which triggered a retraining of the network.

In order to accelerate the simulations, a small percentage $(1-5\%)$ of the connections was therefore removed, as especially the bigger neural networks require a considerable simulation time.

Remark 3: as the primary aim of this research is to find networks of minimal size, no pruning was done before the network had converged. Other researchers regard pruning also as a possibility to speed up learning and remove units before the network converged [Finnoff-93] [Prechelt-95], taking the risk to find neural networks of a sub-optimal size or generalization (in the opinion of the author).

5.3.1 Minimizing Network Size

As the primary aim of pruning is the construction of small networks, pruning methods are usually compared by means of the average final network size.

But this measure neglects that in real applications the training time may or may not be important. Examples are applications for which the final network size is crucial, and the total training time allows the performance of several training sessions. Then, instead of the average network size, the percentage of networks close to an optimal size is important, as a high average can also indicate that some training sessions ended up with oversized networks. The other extreme is that the training of a neural network is very time consuming and can not be repeated often. In this case, rather than the average network size of the percentage of almost optimal neural networks, the amount of largely oversized networks is of interest.

An example justifying this consideration is the outcome of the experiment with a 4th order network applied on the CES data set (see figure 5.2; the horizontal axis shows the

[3]This limit was conservatively chosen, to ensure that the minimal network found could not be pruned any further.

number of connections in the final networks and the vertical axis the percentage of simulations with final networks of this size). Both the smallest weight removal method and the smallest contribution variance method result in networks of an average size of 7.6 connections. Whilst the smallest weight removal method reached in more than 10% of the simulations a final network size of 4 connections, the smallest contribution variance method never produced a network with less then 5 connections (an experiment consists of 200 simulations). A second example is an experiment performed with the Monk 2 data set and a second order perceptron, see figure 5.3: the best result for the smallest weight removal method is a network with 40 connections (in 21% of 100 simulations), and 30 (in 23% of 100 simulations) for the $min(\sigma)$ method.

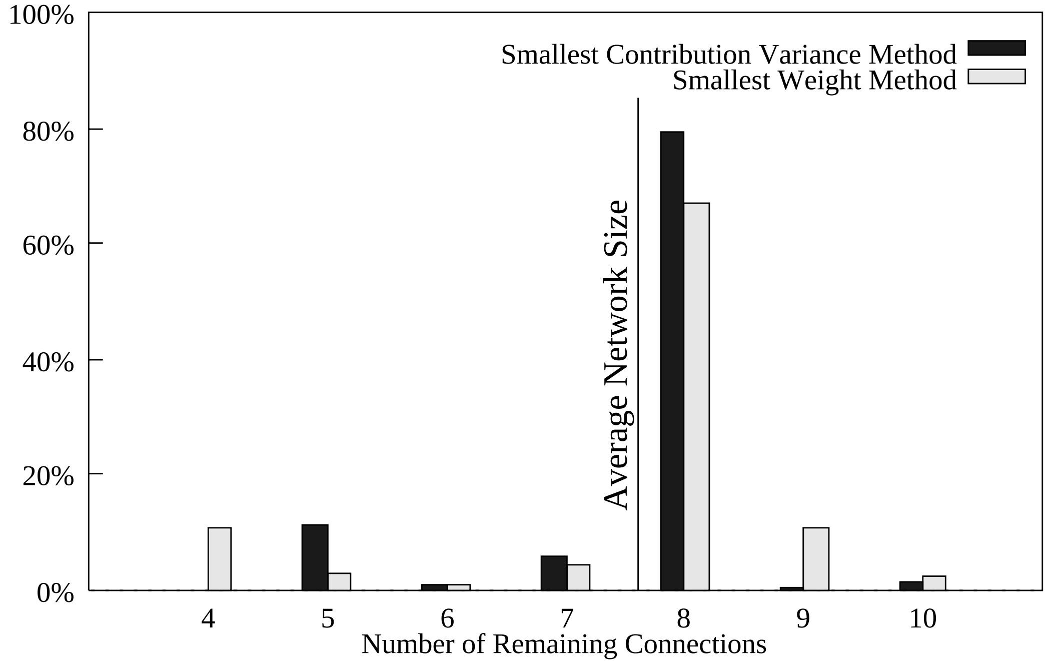


Figure 5.2: The distribution of the final network sizes for the CES data set.

The following performance measure takes these observations into account:

Definition 5.3.1 *(10% and 90% limit for network size)*
The 10% limit, respectively the 90% limit, is an upper limit for the network size which is reached in at least 10%, respectively in at least 90%, of the simulations per experiment.

The network size in the definition can be, for example, the final network size or the size of the network with the best generalization.

Example: in figure 5.2 it can be seen that about 10% of the final networks pruned with the $min(\mathbf{w})$ method have 4 connections, and the 10% limit is therefore 4. As the $min(\sigma)$

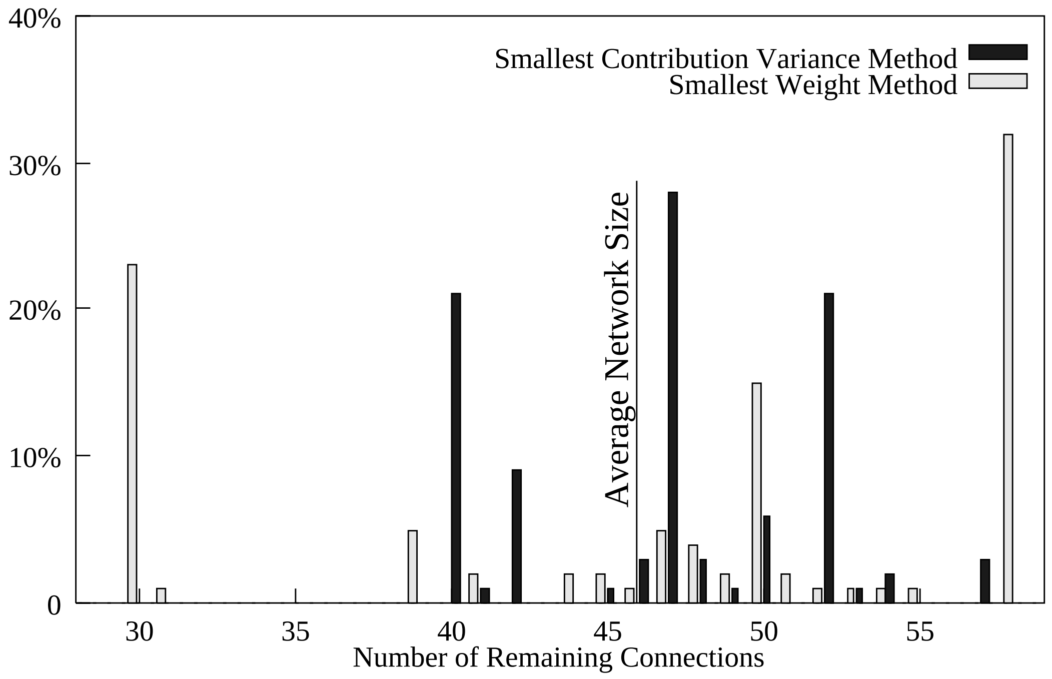


Figure 5.3: The distribution of the final network sizes for the Monk 2 data set.

method never produces networks of 4 connections but a reasonable amount of networks of size 5, the 10% limit is 5. Similarly, the 90% limits differ (they are 9, respectively 8).

Using this measure, method A is judged better than method B, if method A has a smaller 10% (90%) limit, or the methods have the same 10% (90%) limit, and method B produces less networks than method A with this maximal size (assumed that the confidence in this difference is at least 99%; compare appendix A).

Depending on whether the 10% or the 90% limit is applied, either the $min(w)$ or the $min(\sigma)$ pruning method appears to be more effective in the example described above. Usually this discrepancy is observed if the average performance of the two methods is similar.

In table 5.2, five methods are compared[4] using the 10% and the 90% limit criteria on several data sets and networks of different order (Ω is the order of the initial network; 2r means that the network is of order two, but not fully connected[5]). The best result for a certain combination of data set, network order, and the comparison criterion is marked in a bold type faces. Equally marked up are results for which the confidence that they are worse could not be rejected with a confidence of at least 95% (compare appendix A). The

[4]Not all networks were pruned using all sensitivity estimators as it became clear during the experiments that these methods don't perform well, and the available computation resources were limited.

[5]In order to reduce the network to a size that permits the performance of the experiments, only those second order connections are used which take both inputs in the same row or the same column of the image. The performance of the resulting network remains considerably high.

exclamation marks show the lines for which the best method using the 10% and the 90% limit measure result in a different judgement. If two methods are very likely to perform equally good in a certain case, respectively the difference is statistically not significant, both outcomes are highlighted by bold face.

			10% limit for method					90% limit for method					
	Ω	η	*min*(σ)	*min*(w)	Karnin	*skelet.*	*autoprune*	*min*(σ)	*min*(w)	Karnin	*skelet.*	*autoprune*	
Solar	2	0.5	**6**	10	18	62	10	**9**	10	24	76	10	
	3	0.5	**6**	8	16	231	8	**10**	11	24	248	11	
	4	0.1	**8**	13	20	—	13	**13**	18	28	—	18	
Wine	2	0.5	19	**18**	53	53	**18**	**24**	**23**	84	134	25	!!
CES	2	0.1	**3**	**3**	4	4	**3**	**4**	**4**	4	5	**4**	
	3	0.1	**5**	6	6	7	6	**6**	**6**	6	8	**6**	!!
	4	0.1	5	**4**	7	10	**5**	**8**	9	10	13	9	!!
Servo	2	0.01	10	**8**	19	86	**8**	10	**9**	27	89	**9**	
	3	0.01	**22**	33	82	—	33	47	**40**	105	—	**40**	!!
Vowels	2	0.5	**52**	59	126	—	—	**69**	78	186	—	—	
Auto-mpg	2	0.03	13	**8**	13	33	**8**	13	**8**	18	35	**8**	
	3	0.03	9	**6**	14	—	**6**	14	**7**	21	—	**7**	
Glass	1	0.005	**5**	6	12	13	6	**5**	6	14	16	6	
	2	0.005	8	**7**	33	52	**7**	11	**11**	42	78	**11**	
	3	0.005	9	**6**	62	—	**6**	15	**11**	86	—	**11**	
Monk 1	2	0.01	**4**	**4**	44	**4**	**4**	**4**	**4**	80	**4**	**4**	
	3	0.01	**4**	**4**	78	—	—	**4**	**4**	137	—	—	
Monk 2	2	0.01	**30**	40	72	122	40	58	**52**	111	128	**52**	!!
	3	0.01	**28**	**27**	113	—	**28**	**35**	**35**	180	—	**35**	
Monk 3	2	0.01	**11**	11	—	—	11	**11**	12	—	—	12	
	3	0.01	**11**	12	67	15	12	18	19	105	26	**17**	!!
Digits	2r	0.005	**119**	**121**	—	—	**121**	**150**	**153**	—	—	**156**	

Table 5.2: Sizes of the smallest networks obtained by pruning n-th order networks in the 10% and 90% limit.

The table shows clearly, that the method of E. D. Karnin and the *skeletonization* method are surprisingly worse than the smallest weight and smallest contribution variance method. The *autoprune* method of W. Finnoff *et al.* and the smallest weight removal methods perform almost equally in all the cases.

5.3.2 Generalization

Over-fitting of neural networks with a static topology is usually prevented by *early stopping* [Weigend-90]. Although this approach seems to be applicable in combination with neural network pruning, it is shown not to have the desired effect: even though the network pruning can improve the generalization, the retraining usually degrades it. This means that two opposite forces influence the neural network behavior. Furthermore, pruning is not formally shown to impose smoothness onto the function represented by a neural network, as it has been done for the regularization approach (see [Girosi-95] for a review of regularization applied to neural networks).

The common belief in the neural network community is that pruning finally will win the race. In order to validate this, suppose that pruning is an efficient regularization technique (in the sense that it imposes smoothness on the function represented by the neural network). Then following observations should be possible:

- The error on a generalization test set should decrease or remain constant each time a pruning - training cycle is completed successfully (taking some small fluctuations into account).
- The generalization of the smallest network found should be (often) the best among all successful retraining steps during one simulation.

An examination of the results of the experiments performed during this study shows that this is not the case. The changes in the generalization during the training are significant and usually non-monotonic. Figure 5.4 on the next page shows three curves, each describing the typical development of the generalization for simulations using the “glass” data set. The vertical axis in figure 5.4 depicts the error on the generalization test set after each successful retraining step, the horizontal axis the number of remaining connections. The initial networks are fully connected second order perceptrons with 121 connections. The connections are pruned using the smallest weight method to an average number of 9. The final networks obtained during the three the simulations are marked by a big black spot.

It can be seen that, especially when a network approaches its final size, the generalization changes unpredictably and greatly. The main tendency is however towards a better performance for two of the examples, namely those depicted by the straight and the dotted curve in figure 5.4 on the facing page. The dashed curve shows that the training and weight removal may not lead to a good generalization for the smallest network, even if intermediate networks performs well (such networks are marked by the gray background in figure 5.4). Also, the generalization of the trained, but unpruned network is not always worse, in a few cases even better than the performance of the smallest network found by the five pruning methods used for this study.

Table 5.3 shows the percentage of simulations per experiment where the smallest network has a generalization which is at least as good as those of bigger networks during the same simulation. From this table it is clear that the observation described in the last paragraph applies to all of the five methods used for this study. A similar observation was made by L. Prechelt for other pruning methods applied on multilayer perceptrons [Prechelt-95]. The

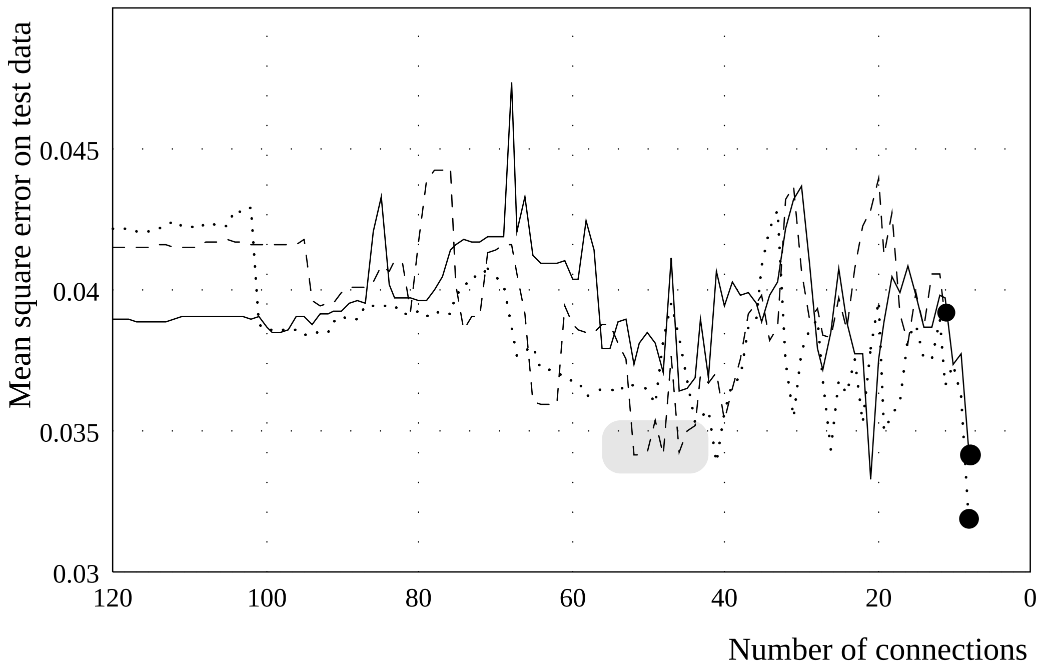


Figure 5.4: The evolution of the generalization during a pruning session.

common belief that the generalization capability of a neural network increases independently from the pruning method and data set with a decreasing size of the network can therefore be rejected.

It follows for practical applications, which require networks of minimal size and good generalization capabilities, that the best intermediate network up to a certain time has to be stored. However, the importance assigned to the generalization and the network size is highly application dependent and therefore no universal halting criterion can be defined. Only two extreme cases are considered in the following sections: the smallest network per simulation and the network with the best generalization per simulation.

Furthermore, as discussed for the network size in the previous section, not the average generalization might be of interest but the generalization obtained in at least 10%, respectively 90%, of the simulations. The corresponding definition is:

Definition 5.3.2 *(10% and 90% limit for generalization)*
The 10% limit, respectively the 90% limit, for the generalization of a neural network is the upper limit for the generalization which is reached in at least 10%, respectively in at least 90%, of the simulations per experiment.

Using this measure, method A is judged better than method B, if method A has a smaller 10% (90%) limit, or the methods have the same 10% (90%) limit, and method B produces less networks than method A with this generalization (assumed that the confidence in this difference is at least 99%; compare appendix A).

	Ω	*min*(σ)	*min*(**w**)	Karnin	*skelet.*	*autoprune*
Solar	2	5	0	2	7	0
	3	5	0	0	–	3
	4	0	0	0	–	1
Wine	2	9	3	11	11	5
Servo	2	6	0	32	0	0
	3	15	26	16	–	25
Auto-mpg	2	0	0	0	90	0
	3	1	0	3	–	0
Glass	1	0	34	19	70	48
	2	1	13	6	2	14
	3	0	25	1	–	22
Monk 1	2	100	100	7	100	100
	3	100	100	4	–	–
Monk 2	2	15	13	10	0	10
	3	59	61	19	–	61
Monk 3	2	39	73	–	–	80
	3	48	55	1	31	58
digits 8x8	<2	0	0	–	–	0

Table 5.3: Percentage of simulations with the smallest network having the best generalization.

Table 5.4 on page 57 shows the generalization for the smallest network per simulation and table 5.5 on page 58 the best generalization per simulation. From these tables it is clear that, similar to the observations made for the final network size, the best method for pruning relative to a data set might well differ for the 10% and the 90% limit criteria. As already observed for the final network size, the method of W. Finnoff *et al.* and the smallest weight removal methods perform almost equally good.

Table 5.5 concerns the generalization of the best generalization per simulation and gives a similar result as table 5.4: the number of experiments, for which the *min*(σ) and the *min*(**w**) methods produced the neural networks with the best generalization, is about equal. But some of the networks produced by the method of E. D. Karnin generalize better than others, and in one case the networks found by the *skeletonization* method do so.

Comparing tables 5.4 and 5.5 for whether the best method differs when looking at the generalization of the smallest network or for the best generalization per simulation, one sees that the best method is in most of the cases equal in the 10% limit but differs often for the 90% limit.

The comparison of the columns with the generalization for the networks, which are not pruned, shows that pruning is in most of the cases beneficial (independent of the pruning method), if the best network is chosen. This is not the case, if the smallest network is chosen. Then in approximately 29 percent of the experiments for which all types of pruning techniques have been applied, the unpruned network generalizes better (for 44 of 152 experiments in table 5.4 the unpruned network performed better).

The tables 5.4 and 5.5 show that, similarly to the observations made for the final network size, the best pruning method can differ for the 10% and the 90% limit criteria. This is the case in about 20% of the series of experiments performed for this study.

All five pruning methods studied perform on average equally well, but a difference in performance on certain data sets exists: networks pruned by the $min(\sigma)$, the $min(w)$, and the method of M. C. Mozer *et al.* classify all test patterns of the monks 1 data set correctly, whereas the method of E. D. Karnin reaches only 88%. For some data sets with a real valued target, the mean square error difference for the networks pruned by the five methods for the test set are as high as 25%.

5.4 Assumptions on the Network Behavior

Although only little of the convergence behavior of neural networks is known *a priori*, one is urged to make assumptions about it in order to have a base for the development of algorithms. Some of these assumptions, albeit they seem to be straight forward and are therefore often exploited in neural network training algorithms, turn out to be invalid. One of those assumptions is the already addressed generalization of a neural network which not necessarily increases with a shrinking network size.

Another assumption is that the generalization first increases with a shrinking network, and when a certain point is reached, it only decreases. A key point in techniques using this assumption is the detection whether the changes in generalization are due to statistical fluctuations or due to the insufficient network size. Those methods therefore keep a record of the generalization progress and the best network during a training session and are stopping the pruning if the generalization degrades for a certain number of connection removals (usually between 3 and 10).

In order to validate this technique, respectively to see which number of connections to remove per pruning step is appropriate to choose, the longest decreasing slope of the generalization per simulation has to be examined. The percentage of occurrence per interval length was calculated for all benchmarks and experiments performed and integrated up to a certain length. The result is displayed in figure 5.5. This plot can be interpreted as a probability to find the network with the optimal generalization if the training is aborted when the generalization decreased for a certain number of removed connections.

Figure 5.5 shows for example, that if the maximal number of removed connections causing a decrease of the generalization is chosen to be 10, the network with an optimal size is found with a probability of 82%. Similarly, if this probability is desired to be at least 90% (95%), this number has to be 22 (47). The longest decreasing slope observed has the length of 226 connections.

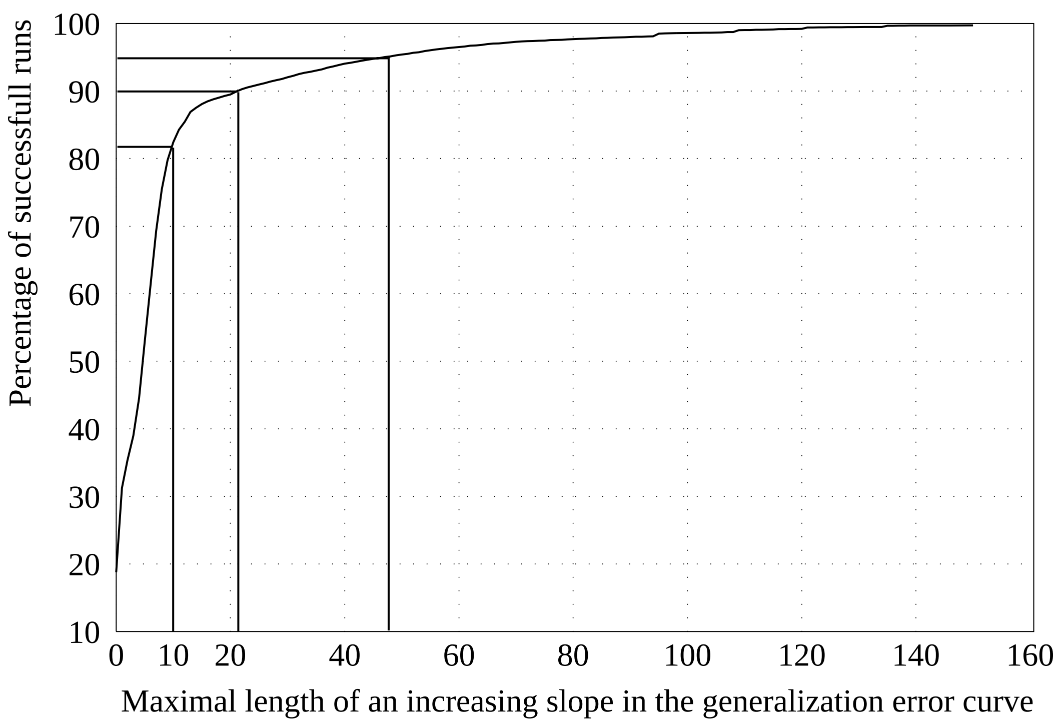


Figure 5.5: Longest decreasing generalization slopes per simulation.

5.5 Complexity of the Methods

A third aspect of pruning methods is the increase in training time. This time splits up into two parts:

- the complexity of the algorithm itself and
- the time needed to retrain the neural network.

Whereas the first item can be calculated for each pruning method, the second is directly related to the quality[6] of the pruning method and therefore inaccessible for a theoretical analysis. Although the complex methods are already excluded, three groups remain: the $min(\mathbf{w})$ method with a complexity $O(C)$, the $min(\sigma)$ method, the methods of E. D. Karnin and M. C. Mozer *et al.* with $O(C*P)$, and the method of W. Finnoff *et al.* with $O(C*P*I)$ (with I the number of training cycles).

That the re-training time can be important is shown in figure 5.6: the average retraining time can differ for two pruning heuristics, as it is demonstrated for a second order high order perceptron trained on the solar data set. After an initial training of about 50 iterations, the first pruning steps require no or only little re-training, as it is expected. Then the

[6]Quality in the sense that the pruned network can be retrained to the same performance as the unpruned network in a low number of iterations.

retraining time increases steadily towards the end of the simulation but not equally fast for both methods: the pruning method of Karnin requires on average more retraining per pruning step. It therefore can be concluded that the method of E. D. Karnin is more likely to remove connections which cause an error than the smallest variance method. This coincides with the observation that the method of E. D. Karnin produces on average bigger networks (the dotted curve is terminated before the straight one). Furthermore, it can be seen that very small neural network are expensive to obtain: the number of retraining steps for small high order perceptrons is much higher than for big ones.

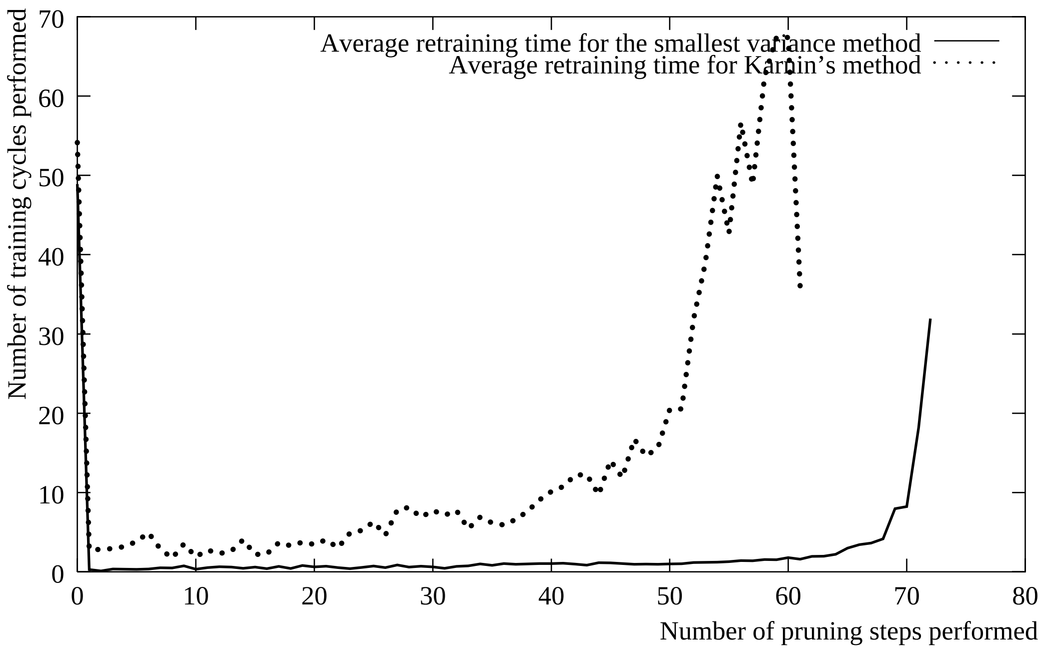


Figure 5.6: The average number of re-training steps versus pruning step.

5.6 Conclusions

I could show that the pruning of higher order perceptrons has sometimes, but definitively not always, the commonly assumed positive influence on the generalization. During a training session, the generalization is not increasing steadily while the network decreases in size but shows an unpredictable behavior. Furthermore, in a few experiments, the unpruned n-th order perceptron generalized on average as good as or even better than the smallest network per simulation. This implies that pruning not necessarily improves the generalization of a network. On the contrary, for some data sets the smallest network almost never has the best generalization per simulation. Therefore, if a network with a good generalization is required, networks of a non-optimal size have to be considered.

The method of W. Finnoff *et al.* can be excluded from the set of potential pruning methods, as it performs almost equally as compared to the smallest weight removal method, both in terms of network size and generalization, but has a higher computational complexity, and its implementation needs more effort.

I found that a significant difference in generalization of the final networks produced by the four remaining pruning methods can not be stated generally, only for specific data sets, where the differences are sometimes remarkable. Consequently, if high order perceptrons with a good generalization are required, several training sessions using different pruning methods are inevitable, as the best pruning method for a specific data set is *a priori* unknown. This observation and the resulting conclusion probably apply also to other neural network architectures.

The method of E. D. Karnin and the *skeletonization* method are less efficient in finding small high order perceptrons than the $min(w)$ and the $min(\sigma)$ method, independent of the measure used. Furthermore, the latter have the advantage of a lower complexity. Only in a few cases these methods produce networks of a size comparable to those found by the $min(w)$ and the $min(\sigma)$ method, more often the final network size is two or more times as large. The difference between the $min(w)$ and the $min(\sigma)$ method is less significant with a maximal difference of 50%. The number of experiments, where one of these methods is shown to be more efficient, are comparable.

Pruning methods using penalty terms are excluded from the experiments, as they can force the training process into a "false" minimum which may or may not be close to the global minimum of the error surface (see section 5.2). Consequently, a neural network should be trained at least in the final stage of the training without a penalty term. In other words, regularization techniques should not be used as a pruning technique, but only in the intension to impose a task related property on the function represented by the trained neural network.

The in the sections 5.3.1 and 5.3.2 introduced 10% and 90% limit measures for the network size and generalization show that the performance of two pruning methods can differ remarkably for a certain data set, although the mean generalization or the mean final network size is equal for both methods. However, no pruning method is *a priori* preferred.

		10% limit for method						90% limit for method						
	ω	Unpruned	*min*(σ)	*min*($\mathbf{w}$)	Karnin	*skelet.*	*autoprune*	Unpruned	*min*(σ)	*min*($\mathbf{w}$)	Karnin	*skelet.*	*autoprune*	
Solar	2	0.078	0.077	0.095	***0.075***	0.089	0.095	0.104	***0.087***	0.103	0.089	0.108	0.103	!!
	3	0.096	0.078	0.087	***0.075***	—	0.084	0.139	0.111	0.105	***0.087***	—	0.105	
	4	0.077	***0.076***	0.077	0.079	—	0.077	0.090	***0.084***	0.092	0.092	—	0.092	
Wine	2	0.122	***0.122***	0.146	***0.122***	***0.122***	***0.122***	0.146	***0.195***	0.220	0.220	***0.195***	***0.195***	!!
Servo	2	0.137	***0.106***	0.123	0.114	0.140	0.123	0.139	***0.110***	0.124	0.124	0.145	0.124	
	3	0.117	0.094	0.105	***0.091***	—	0.105	0.119	0.121	0.116	***0.111***	—	0.116	
Auto-	2	0.055	0.058	0.059	0.058	***0.054***	0.059	0.056	0.059	0.061	0.061	***0.056***	0.061	
mpg	3	0.056	0.058	0.060	***0.057***	—	0.060	0.059	***0.061***	0.062	0.062	—	0.062	!!
Glass	1	0.032	0.033	0.028	0.032	***0.027***	0.028	0.034	0.036	0.032	0.035	***0.028***	0.032	
	2	0.038	0.041	***0.034***	0.037	0.036	***0.034***	0.043	0.046	***0.041***	0.046	0.048	***0.041***	
	3	0.043	0.046	***0.034***	—	—	***0.035***	0.050	0.051	***0.044***	—	—	***0.044***	
Monk 1	2	0.201	***0.000***	***0.000***	0.123	***0.000***	***0.000***	0.218	***0.000***	***0.000***	0.222	***0.000***	***0.000***	
	3	0.275	***0.000***	***0.000***	—	—	—	0.315	***0.000***	***0.000***	—	—		
Monk 2	2	0.215	***0.046***	0.093	0.120	0.160	0.093	0.222	***0.116***	0.125	0.206	0.192	0.127	
	3	0.368	***0.160***	***0.169***	0.269	—	***0.169***	0.389	***0.269***	***0.269***	0.326	—	***0.266***	
Monk 3	2	0.097	***0.097***	0.125	—	—	0.123	0.175	***0.144***	0.176	—	—	0.176	
	3	0.074	0.083	***0.074***	0.236	***0.074***	***0.074***	0.234	***0.139***	0.148	0.368	0.167	***0.134***	!!
Digits 8x8	2r	0.076	0.078	***0.074***	—	—	***0.074***	0.100	0.104	***0.098***	—	—	***0.098***	

Table 5.4: The error on the test set of the smallest network per pruning session in the 10% and the 90% limits.

		10% limit for method						90% limit for method						
	ω	Unpruned	$min(\sigma)$	$min(\mathbf{w})$	Karnin	*skelet.*	*autoprune*	Unpruned	$min(\sigma)$	$min(\mathbf{w})$	Karnin	*skelet.*	*autoprune*	
Solar	2	0.078	0.070	0.070	***0.066***	0.075	0.070	0.104	0.078	0.081	***0.075***	0.092	0.080	
	3	0.096	0.072	0.073	***0.064***	0.094	0.072	0.139	0.082	0.086	***0.073***	0.111	0.083	
	4	0.077	***0.069***	0.070	***0.069***	—	0.070	0.090	***0.072***	0.074	0.073	—	0.074	
Wine	2	0.122	0.098	***0.098***	0.098	0.098	***0.098***	0.146	***0.122***	***0.122***	0.122	0.122	***0.122***	!!
Servo	2	0.137	***0.104***	0.112	0.114	0.137	0.111	0.139	***0.106***	0.114	0.119	0.139	0.114	
	3	0.117	0.092	0.104	***0.091***	—	0.104	0.119	0.109	0.115	***0.106***	—	0.115	
Auto-mpg	2	0.055	0.055	0.055	0.055	***0.054***	0.055	0.056	0.056	***0.056***	0.056	0.056	***0.056***	!!
	3	0.056	0.056	***0.054***	0.056	—	***0.054***	0.059	0.058	***0.056***	0.057	—	***0.056***	
Glass	1	0.032	0.031	0.028	0.032	***0.026***	0.028	0.034	0.032	0.029	0.034	***0.028***	0.030	
	2	0.038	0.037	***0.033***	0.036	0.034	***0.033***	0.043	0.039	***0.036***	0.039	***0.038***	***0.036***	
	3	0.043	0.039	***0.034***	0.038	—	***0.034***	0.050	0.043	***0.039***	0.043	—	***0.039***	
Monk 1	2	0.201	***0***	***0***	0.069	***0***	***0***	0.218	***0***	***0***	0.150	***0***	***0***	
	3	0.275	***0***	***0***	0.213	—	—	0.315	***0***	***0***	0.285	—	—	
Monk 2	2	0.215	***0.019***	0.083	0.104	0.139	0.083	0.222	0.109	***0.102***	0.174	0.141	***0.104***	!!
	3	0.368	***0.157***	?0.169	0.257	—	***0.164***	0.389	***0.259***	***0.262***	0.315	—	***0.262***	
Monk 3	2	0.097	***0.097***	0.125	—	—	0.120	0.175	***0.130***	0.160	—	—	0.160	
	3	0.074	0.079	***0.074***	0.201	***0.074***	***0.074***	0.234	***0.120***	0.125	0.259	0.130	***0.120***	!!
digits 8x8	2r	0.076	***0.048***	***0.048***	—	—	***0.046***	0.100	0.054	***0.054***	—	—	0.054	

Table 5.5: The error on the test set of the best performing network per pruning session in the 10% and the 90% limits.

Chapter 6

Incremental Growing

As already stated in chapter 5, pruning methods, which are able to find small and problem suiting neural network topologies, have been developed. Although these methods are rather efficient, they have the drawback that the initial network size has to be guessed. Therefore this approach may involve a long trial-and-error process: the number of trials is often further increased as the user has no clue about adequate values for the various learning parameters. It easily happens then that users change the network topology, while the poorly chosen learning rate is the real cause for non-convergence. Obviously this can not be avoided, as it is unclear whether the network does not learn due to an unsuiting topology or an unlucky choice of learning parameters.

This problem can be partly overcome by a directed search for a suitable network topology. A possible approach starts a training session with a network that is too small and to adapt its topology to the problem by adding units during the training process. This has already been done for various neural network architectures (see section 6.1.1) and even for high order perceptrons (see section 6.1.2). However, the latter approaches are restricted to Boolean data or classification problems, whereas no method with a limited complexity, that is furthermore applicable to high order perceptrons and regression problems[1], is known.

The choice of the connection(s) to add into high order perceptrons is crucial for the success of such a growing approach. All connections are distinct in their function and, as opposed to neurons in multilayer perceptrons, not interchangeable: the set of all products (of the inputs) forms an orthogonal set of functions. This implies that no connection can be (exactly) replaced by any linear combination of other connections. Or, it is impossible for a high order connection to change the feature[2] for which it is sensitive; the training of a high order perceptron changes only its importance on the output value by modifying

[1]Regression is meant in the statistical sense: the difference between the network output and the target value(s) is subject to minimization (in contrast to classification problems).

[2]A feature (in the input vectors of a pattern set) is a typical property of the data that is useful for the performance of the given task.

the corresponding weight. In contrast, the function set, which is defined by the hidden neurons of a multilayer perceptron, is not orthogonal. Obviously, these neurons can change the feature(s) for which they give a response, and their responses are interchanged together with their weight vectors. It is further imaginable that a set of hidden neurons take over and replace without any lost the function of a neuron that is or will be removed.

In consequence to the non-interchangeability of high order connections, users without any guidelines are anxious not to add (to "forget") important ones. If the training of a high order perceptron failed, they are tempted to add connections order-wise. Accordingly, the network grows exponentially to the number of inputs from trial to trial. The growth rate is especially excessive for data sets with a big number of input elements, as for example in image recognition. For such data sets, second order perceptrons are already oversized, and both the memory occupied by the network and the training time are cumbersome. It is therefore important to construct networks of a reasonable size which are able to correctly learn a data set.

After a literature overview on the different growing approaches for neural networks, this chapter will outline and analyse an unsuccessful approach to grow high order perceptrons *incrementally*, which means by adding from time to time a few connections. The approach called *constructive growing* in chapter 7 works differently: each time a network does not learn, an entirely new and bigger one is constructed.

The author hopes that the reported deficiencies of this approach help other researchers to avoid the encountered pitfalls and to understand some design decisions of the more successful constructive algorithm described in chapter 7.

6.1 Literature Overview on Growing Methods

Earlier overviews on neural network growing methods were carried out by T.-Y. Kwok and D.-Y. Yeung [Kwok-95], as well as by E. Fiesler [Fiesler-96]. However, due to the continuous surge of new methods the referenced overviews are no longer complete. In the following section, a list of some of the missing methods is given, while those applicable to high order perceptrons can be found in subsection 6.1.2.

6.1.1 Literature Overview on Growing Methods for Miscellaneous Neural Network Topologies

Several researchers developed methods based on Boolean logic, which, as already discussed, have the disadvantage of being restricted to classification problems in the Boolean domain. These methods can be found in: [Beiu-95], [Fahner-92], [Fahner-94], [Gray-92], [Kowalczyk-94], [Madineni-94], [Mascioli-93], [Parsons-92], [Raina-94], [Redding-93], [Campell-95], and [Jackson-96].

This restriction to classification problems is not present in other approaches. S. Shiotani *et al.*, for example, grow radial basis multilayer perceptrons by adding new hidden layer neurons [Shiotani-95]. Likewise, J. C. Platt either trains a radial basis multilayer perceptron when a new pattern is presented or, when the latter is estimated to be insufficient, adds a further hidden layer neuron memorizing the new pattern [Platt-91]. Another algorithm, which increases the size of the hidden layer(s), is described in [Bodenhausen-91]. This approach is based on the idea of splitting neurons with big per-pattern calculated weight changes. Similarly, Ö. Ekeberg and A. Lansner use correlations to find higher order dependencies and replace two internal units by three [Ekeberg-89].

A multilayer perceptron with pair-wise orthogonal activation functions in the neurons of the single hidden layer is described by V. Vyšniauskas *et al.* The growing algorithm, which adds hidden neurons, profits from the orthogonality of the hidden layer activation functions [Vyšniauskas-95].

A method not restricted to a certain type of neural networks was presented by E. B. Bartlett. Neurons with random weights are added and the network pruned alternately, until the network appears to be in a stable state [Bartlett-94]. A more recent method for growing multilayer perceptrons layer-wise is described by J. O. Moody and P. J. Antsaklis. In this method layers are added subsequently until the output layer gives a correct response [Moody-96]. A growing algorithm for multilayer perceptron-like network with a non-uniform topology was developed by V.V. Vinod and S. Ghose [Vinod-96].

A method similar to genetic algorithms was proposed by K. Mohraz *et al.* [Mohraz-96].

J.-Y. Choi *et al.* add hidden layer neurons to a perceptron with the weights being calculated by a method developed for system identification (the connections of the former perceptron become supralayer connections). Only the new weights are trained with the backpropagation algorithm [Choi-96].

Some growing methods are based on modifications of well known types of neural networks: B.-L. Lu *et al.* add new modules to a modular, multi-sieving neural network [Lu-95]. B.-L. Lu and K. Ito further developed this architecture by adding so-called *multiple control networks*, which improve the decomposition in sub-learning tasks [Lu-96]. N. Burgess *et al.* propose a mixture of the Cascade Correlation and Upstart algorithm [Burgess-92]. C. L. Giles *et al.* extended the Cascade Correlation algorithm [Fahlman-90] to recurrent networks [Giles-95], and L. R. Leerink *et al.* generalize it by introducing product[3] and cosine units [Leerink-95]. R. Parekh *et al.* proposed an extension of the Pyramid Networks, as well as the Upstart, the Tower, the Cascade, and the Tiling to multi-category problems [Parekh-96].

C. M. Higgins and R. M. Goodman interpret a network as a kind of expert system. Their algorithm starts with a set of high order rules, which are determined from the data set, and tries to simplify it by replacing complicated rules with lower order ones. Finally, each rule is translated into a neuron which are assembled into a network [Higgins-91].

An interesting idea, which is applicable to different network architectures, was presented by T. M. Nabhan and A. Y. Zomaya. Their decision about which neuron or layer to insert relies on weight changes during the training [Nabhan-94].

Algorithms based on a division of the input space into different parts are explained in [Sankar-91], [Shadafan-93], and [Lee-92]. These algorithms are not applicable to high order perceptrons as they are based on the idea that each hidden layer neuron can be regarded as a hyperplane dividing the input space into two parts. A related approach was done

[3] A neuron has the form $\Pi_i x_i^{w_i}$

by J. H. Friedman, who unified k-nearest neighbor and decision tree induction algorithms [Friedman-96]. An overview and extension to a few instance-based learning algorithms, which are much like nearest neighbor algorithms, to noise-tolerant methods was proposed by D.W. Aha and D. Kibler [Aha-89].

6.1.2 Literature Overview on Growing Methods for High Order Perceptrons

Some methods for growing high order perceptrons have already been developed: M. Ring proposed an algorithm that adds neural layers with first and second order connections [Ring-93]. However, all connections between two layers are added at once; the critique about the size of the network in the introduction to this chapter applies.

The algorithm described in [Sanger-91] sequentially adds connections to high order perceptrons by considering high order terms which are built by multiplying already existing ones with one further input value. The estimation for the term to add is done by three regressions, which are aimed at finding connections that minimize the error as well as to determine the corresponding weights.

A few methods for growing high order multilayer perceptrons are known: M. F. Tenorio and W.-T. Lee add second order polynomials to a tree-like structure. Their method shows a strong resemblance to genetic algorithms [Tenorio-90], thus having the disadvantage of a considerable demand for calculation power. M. I. Heywood and P. D. Noakes construct high order multilayer networks for Boolean problems using the correlation of a term with the error local to a neuron [Heywood-94]. This algorithm is not suitable for regressions, and the selection of connections is rather expensive in terms of computation time.

6.2 An Approach to Growing High Order Perceptrons

This approach is divided into two parts: in a first phase, the prospective connections are ordered according to their estimated information content in the network. The set of prospective connections usually contains all possible first and second order ones. If they are insufficient to solve the problem, successively connections of increased order are included. The estimation of the information content assigns only a relative importance to the connections (i.e. defines an order among the connections). The estimation of the absolute error caused by the absence of a connection in a network would require methods like principal component analysis which are computationally very expensive. It is therefore not compatible with the intention of saving computing resources as compared to the pruning approach, being even impossible for big data sets. Furthermore, the re-estimation of the importance of a connection is done each time a network is grown. The required computational effort for finding a good network will then be similar or higher, as compared to the pruning approach discussed in chapter 5, because it requires a sweep over the whole set of prospective connections. However, the pruning approach is likely to produce better networks, as additional information on whether a certain connection is useful or not can be gained from the weight attached to it. Consequently, growing is justified only if it has a lower computational complexity as compared to the pruning approach.

The first heuristic is defined in the following way: a connection c_i has more importance than c_k if the output variance

$$\sigma_p(c_i) = \frac{\sum_{p=1}^{P} (c_i^p - \overline{c_i})^2}{P-1}$$

calculated over the training set for c_i is bigger than for c_k:

$$\sigma_p(c_i) > \sigma_p(c_k).$$

The second heuristic is based on the covariance of the output of the connections with the target: if

$$cov_p(c_i, t) > cov_p(c_k, t),$$

c_i has more importance than c_k, where $cov_p(c, t)$ is the covariance defined as

$$cov_p(c, t) = \frac{\sum_{p=1}^{P} (c^p - \bar{c}^p)(t^p - \bar{t}^p)}{P-1}.$$

The growing of the high order perceptron is performed in the second phase of the algorithm. In this phase, connections are added one by one, those with a high estimated importance first, whenever the network was unsuccessfully trained for a certain number of training steps.

6.3 Description of the Experiments

During these experiments, the neural networks are trained with the same parameters as in chapter 5. The weights of the newly inserted connections are set to random values. The number of training steps is conservatively chosen[4], in order to permit convergence of the networks.

Table 6.1 shows the average network size obtained during the experiments with four data sets (each experiment consists of 50 simulations). In this and the following tables the notation $A \pm B$ indicates that the 95% t-student confidence interval is (not bigger than) $[A-B, A+B]$; $\sigma(c_i)$ means that the variance method was used to grow the network, $cov(c_i)$ the covariance method, and *rand.* the random insertion of connections. Columns with $\Omega = 2$ show the sizes of (un-)pruned fully connected second order perceptrons.

It can be seen that, except for the wine data, growing is stopped before the size of a fully connected second order network is achieved. However, the final, pruned network remains often bigger than the pruned second order networks.

An outstanding observation in these experiments is the big difference between the grown but unpruned, and the final network. On the reason for this difference can be speculated.

[4]The number of training steps was chosen to be up to 10 times as much as necessary for a fully connected second order network to converge.

	Size of unpruned networks				Size of pruned networks			
	$\sigma(c_i)$	$cov(c_i)$	rand.	$\Omega=2$	$\sigma(c_i)$	$cov(c_i)$	rand.	$\Omega=2$
Wine[†]	93±6	382±3	265±86	274	22±0	35±2	49±3	20±1
Solar	18±1	16±0	41±3	78	7±0	13±0	19±2	7±0
Glass	52±0	12±1	27±3	121	25±2	8±1	14±2	9±0
Servo	35±1	23±1	48±3	91	18±2	17±1	21±2	8±0

[†] All third order connections are used, as simulations using only second order ones often failed to converge!

Table 6.1: Comparison of network sizes obtained by the growing-pruning methods and pruning of fully connected second order networks.

Besides a possible bad selection of the candidate connections, this may also be caused by weights which "drifted" far away from the global minimum in the error surface (during the long training phases). Therefore similar experiments have been performed with the difference that the weights are randomized whenever a new connection is added to the network. The corresponding results are shown in table 6.2.

A comparison of tables 6.2 and 6.1 shows that the deterioration of the weights is not the main cause for the networks being grow beyond the required size: except for the wine data, the differences between table 6.1 and 6.2 are small, and for the servo data set even the maximal network size is increased. However, the difference between inserting randomly chosen connections and the two proposed methods justifies their consideration.

	Size of unpruned networks			Size of pruned networks		
	$\sigma(c_i)$	$cov(c_i)$	rand.	$\sigma(c_i)$	$cov(c_i)$	rand.
Wine	49±1	162±3	63±1	23±0	47±3	32±1
Solar	18±1	16±0	39±3	9±1	10±1	18±2
Glass	51±0	12±0	23±0	13±1	6±0	12±0
Servo	40±1	29±1	40±1	13±1	10±1	13±1

Table 6.2: Comparison of network sizes obtained by the growing-pruning methods with intermediate weight randomization.

The comparison between tables 6.1 and 6.2 shows that the randomization of the weights is only partially successful, namely for the wine data, but it increases the intermediate network size for the servo data set.

In tables 6.3 and 6.4, the average of the smallest error on the generalization test sets per simulation are given together with the same figures obtained during the pruning experiments performed for chapter 5.

The comparison between the generalization errors shows that, except for the glass data set and the covariance method, the random insertion of connections produce networks of inferior generalization capabilities (with or without weight re-randomization). The perform-

	Average generalization error			
	$\sigma(c_i)$	$cov(c_i)$	rand.	$\Omega=2$
Wine	12.9±5%	13.6±4%	19.7±16%	9.9±1%
Solar	0.070±1	0.079±0	0.083±3	0.075±1
Glass	0.041±1	0.028±0	0.036±1	0.034±0
Servo	0.115±1	0.114±1	0.129±4	0.105±0

Errors are in percent misclassification, respectively in mean square difference

Table 6.3: Comparison of the generalization of growing methods without intermediate weight randomization.

	Average generalization error		
	$\sigma(c_i)$	$cov(c_i)$	rand.
Wine	16.3±4%	14.5±6%	19.5±0%
Solar	0.072±2	0.075±1	0.084±3
Glass	0.037±1	0.033±1	0.030±0
Servo	0.111±1	0.114±2	0.120±1

Errors are in percent misclassification, respectively in mean square difference

Table 6.4: Comparison of the generalization of growing methods with intermediate weight randomization.

ance of the pruned second order networks is sometimes worse but mostly better than the growing-pruning methods with unchanged weights. If they are re-randomized, the resulting networks perform worse or equally well, except for those trained with the $\sigma(c_i)$ method on the solar data set. Comparing the random insertion of connections to the other methods, these perform usually but not always better.

The required training time using the growing methods is much higher (estimated 10 to 50 times) than for pruning a fully connected second order network. A less conservatively chosen number of training steps before a connection is inserted could reduce the training time, but unlikely below the time for the pruning method.

6.4 Discussion of the Experiments

The outcome of the experiments in section 6.3 clearly shows the following shortcomings of the described approach, independently of the use of the two heuristics for the addition of connections:

1. During a simulation, the networks tend to grow until they are more than twice as big as the final network (compare the left and the right parts of the tables 6.1 and 6.2).

2. If the grown, converged networks are pruned, their sizes remain often bigger than of those found during the experiments where fully connected second order networks are pruned (a maximal factor of 2.78 was observed).

3. The difference of the size to which the networks grow using the two heuristics described above as compared to when adding randomly selected connections is rather small.

4. The generalization capabilities of the best network per simulation are either worse or better than those obtained by pruning fully connected second order networks.

The first three observations in the list above lead to the conclusion that the *a priory* estimations used for growing are less efficient than the *a posteriori* criteria used in the pruning methods presented in chapter 5. This is not really surprising as pruning profits from the additional information contained in the weights. Growing methods are therefore more likely to insert "wrong" connections than a pruning method to remove "good" ones.

It appears that during the training of the networks of an insufficient size, the weights tend to assume values which are far off the global minimum. This cause of late convergence can be reduced by randomizing all weights after a connection is added to the high order perceptron. This reduces the maximal intermediate network size but not to the desired amount and, for some data sets, on the cost of a reduced generalization.

The unreasonably high training time is caused by two deficiencies of this approach: first, the decision whether a network will converge or not is delicate. An algorithm that observes the progress of the training more closely, i.e. the slope of the error curve and its distance to the accepted error, can help this. The disadvantage of such algorithms is the introduction of more, difficult to tune learning parameters, as it can be observed in similar approaches used with pruning algorithms (for example in [Prechelt-95], [Heywood-93], and [Beuchat-95]). Second, the introduction of only one connection at a time is inefficient. However, there is no indication of how many connections are missing. Consequently, at least one additional user defined learning parameter is required.

6.5 Conclusion

It can be concluded that, although the methods construct intermediate networks of smaller size than those of fully connected second order ones, the results are not completely satisfying: the training time is considerably bigger, the network size still far from optimal, and the users are hampered by some extra parameters. Furthermore, the problem of an exponentially growing number of connections is not entirely solved: if connections up to the n-th order are considered, then the importance has to be calculated for each of them.

It follows that a better growing method should not estimate the importance of connections in a pre-selected set but directly indicate which connection(s) to add. It is important that several connections are added during each growing phase in order to keep the training time within reasonable limits. Ideally the number of connections should not be specified by the users but emerge from the method.

Chapter 7

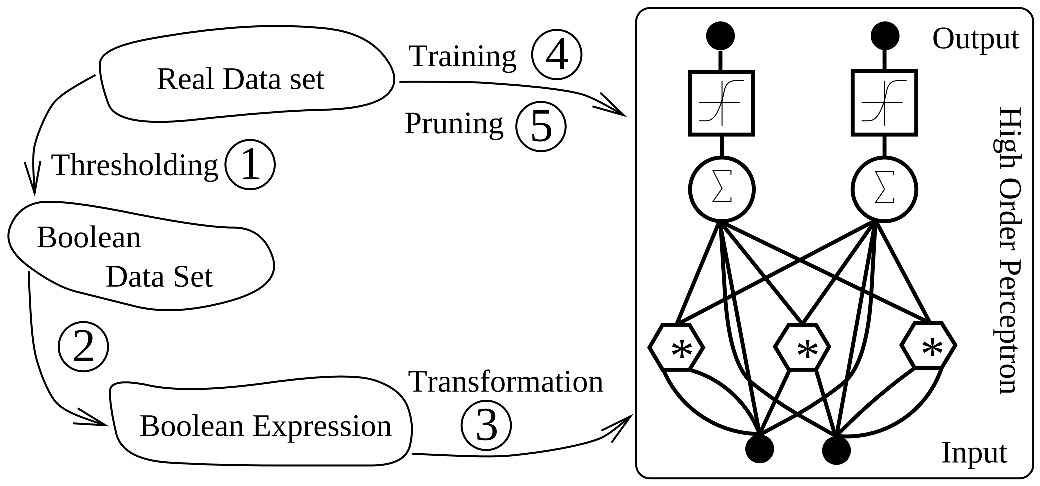
Constructive Growing

7.1 The Approach

The construction of higher order perceptrons or other types of neural networks for Boolean problems can be regarded as being solved since many algorithms are known. The main subject of the ongoing research in this domain is the optimization of these methods, see for example [Crama-88], [Gray-92], [Fahner-92], [Wu-92], [Pol-95], and [Beiu-95]. The networks produced by the algorithms presented in these papers differ mainly in properties like generalization, topology, weight ranges, *etc.* Unfortunately, these algorithms take advantage of the data being Boolean and are therefore not directly applicable to many real-world data.

A straight forward and often referenced technique to make these algorithms applicable to non-Boolean data is to discretize the real-valued data and to code each continuous-valued element by a Boolean vector. This approach has two major disadvantages: the networks constructed have a significantly larger size as compared to those acting on the original data since more in- and outputs are necessary, and therefore also the probability for a poor generalization performance increases. Besides, in contradiction to one of the aims of this dissertation, further parameters are introduced: the number of discretization levels and their values.

A "human" way to construct a network is to search a small set of quasi-logic rules which explains the basic relations between input and target vectors. These rules are most likely of the form **if value A is big and value B is small then the output is big**. The developer first applied a sort of fuzzy threshold function and tried then to find a set of rules based on this approximation. When a set of rules, that explains most of the data, is found, these rules are quantified and finally refined. The "machine-like" method inspired by these thoughts consists of five basic steps which are illustrated and described in figure 7.1. It is an important feature of this approach that the Boolean data is exclusively involved in the construction phase; during the training and pruning phase only the original data set is used.



① the real- or discrete-valued patterns are transformed into a Boolean data set by thresholding the elements of the input, respectively output vectors (one element in the original data set is transformed in one element in the Boolean data set).

② This Boolean data set is used to construct a Boolean expression which

③ is translated into a high order perceptron.

④ As a network constructed for the Boolean data will perform poorly on the original, real valued data set, this network is trained using the backpropagation algorithm and

⑤ finally pruned.

Figure 7.1: The construction and training scheme.

Note that this approach is inherently unable to cope with all data sets: the Boolean approximation (at least in the way in which it is done here) is not always able to extract the relevant knowledge from the data set (see section 7.3 for more explanations).

① Thresholding the Real-Valued Data Set

In this first step, the continuous or discrete valued data are thresholded in the aim to obtain a Boolean data set which has as much similarities with the original data set as possible. The threshold value is chosen to be the mean of each input element ξ_i (similar for the targets t_j^p):

$$b_i^p = \begin{cases} true & if\ \xi_i^p > \sum_p \xi_k^p / N \\ false & \text{otherwise} \end{cases}$$

This threshold value is probably close the intuitively used threshold when a human tries to find clues on which connections to choose. However, the median has a higher probability to create different Boolean vectors for different input vectors: assumed that the input elements are statistically independent, the average number of original data mapped onto the same Boolean vector is approximately equal to the size of the data set divided by two to the power of the size of the input vector. Both threshold values are not necessarily optimal in terms of creating the maximal number of different Boolean vectors for a specific data set, but an optimal value is expensive to calculate[1]. Changing this threshold will influence the resulting Boolean function.

It occasionally happens that different input vectors of the original data set are mapped onto the same Boolean vector but not their target vectors. Therefore the obtained Boolean data set not necessarily defines a function. In order to obtain a sound Boolean data set[2], one of each pair of those contradicting elements is omitted from the data set which is used for the calculation of the Boolean expression. For this research, the patterns, for which the target vector is most contradictory to the patterns with similar inputs (i.e. those having the smallest Hamming distance), are omitted from the Boolean data set. Note that the original data set remains unchanged, and there is no reason why the data corresponding to the omitted Boolean patterns are not learnable by the constructed network.

Omitting the contradictory data is preferred over omitting non-contradictory data, as this choice promises smaller networks with a smoother function surface. However, the influence of this choice on the final topology of the network can be estimated to be inferior to changing the threshold values.

Example: Assume the data set consists of the following patterns:

ξ_1	ξ_2	ξ_3	$\rightarrow t$
0.6	1.0	0.9	$\rightarrow$ 0.8
0.4	0.6	0.2	$\rightarrow$ 1.0
0.9	0.0	0.1	$\rightarrow$ 0.6
0.1	0.8	0.8	$\rightarrow$ 0.0

Then the respective thresholds are:

$$0.5 \quad 0.6 \quad 0.5 \rightarrow 0.6$$

Applying these thresholds to the data set results in the following Boolean data set:

b_1	b_2	b_3	$\rightarrow$ t	Pattern number
true	**true**	**true**	$\rightarrow$ **true**	**A**
false	**true**	**false**	$\rightarrow$ **true**	**B**
true	**false**	**false**	$\rightarrow$ **true**	**C**
false	**true**	**true**	$\rightarrow$ **false**	**D**

As easily verified, no contradictory data is contained in this set.

[1]This optimal threshold value is subject to future research.

[2]A Boolean data set is considered to be sound if it defines a function, i.e. does not contain two data with equal input but different target vectors

② Calculation of the Boolean Expression

In this step, the data set is converted into an analytical function. This is an important step, as the transformation of a Boolean expression into a high order perceptron is almost syntactically and hides no further problems. In a first implementation of this approach, which uses disjunctive normal forms, has promising results [Pol-95] but also some drawbacks:

- the networks constructed from this Boolean expressions are much bigger than necessary (sometimes bigger than the corresponding fully connected second order network),
- They often contain many connections of a high order (i.e. bigger than four) but only a few of a lower order, and
- therefore have a poor generalization performance.

It is evident that this approach can be improved by discarding connections of an excessively high order. Unfortunately, this attempt often fails as often lower order connections are missing. A. de Pol et al. did a step towards solving this problem: the Boolean expressions are crudely simplified by eliminating the negated variables. This reduces in many cases the number of literals per conjunction and consequently the order of the corresponding connections, but the same problem persists (compare [Pol-95]).

An algorithm which allows the calculation of Boolean expressions with better properties for the presented method than those of disjunctive normal forms is therefore necessary.

An algorithm suitable for this approach was initially developed by Y. Crama *et al.* [Crama-88] and further refined, as well as implemented by E. Boros *et al.* [Boros-96]. This program offers, besides being optimized in calculation time, some useful features which allow tuning the size of the initial network in a more reasonable way than simply increasing its order:

- The order of the terms can be hard limited, but connections of higher order are not simply omitted. The construction algorithm takes the interdiction of their use into account.
- The number and order of connections can be reduced at the cost of misclassifications: the extraction of the minimal set of basic pattern[3] may be done in a way that only a percentage of training patterns are well-classified.
- It can be specified that each basic patterns is true for at least a certain number of data patterns. This reduces the set of searched basic patterns and can therefore be expected to increase the size of the constructed networks. Furthermore, the higher this parameter, the more likely it is that the network generalizes well.

[3]A basic pattern is a conjunction of some Boolean variables or their negations, which is true for some data with one type of output and false for any data with a different output.

- It can be specified that for each input pattern a certain number of basic patterns is true (the standard case is one). This parameter can be used to increase the size of the network in a sensible way.
- The Boolean expression is optimized by using the don't-know terms that emerge implicitly from Boolean vectors onto which no data element is mapped during step ①.

Example: among the Boolean patterns in the example of the last paragraph are, in the terminology of LAD, three positive patterns (those with target value **true**) and one negative pattern (with target value **false**). Using a coverage of 2, LAD finds that the Boolean term b_1 is true for the positive patterns **A** and **C** and false for all negative patterns. Equally, the Boolean term $\bar{b}_3$ is only true for the positive patterns **B** and **C**. Whereas the term $\bar{b}_1 \wedge b_3$ is only true for the negative pattern **D**.

③ Creation of the network

In this phase the Boolean expressions (respectively the sets of basic patterns) found for each output are transformed into polynomials. A Boolean expression can be translated into a polynomial of the corresponding real-valued variables and coefficients. The latter can be calculated from the precise form of the conjunction [Rumelhart-86]. This calculation is not done here, for three reasons:

- In consideration of the crude approximation of step ①.
- As a sigmoidal function is applied to the output of each polynomial, it is likely that some terms are superfluous (higher order terms can be pruned in the presence of lower order terms).
- In order to save computation time required for the calculation of these terms.

More precisely, each basic pattern corresponding to a certain output is translated into a polynomial with all combinations of possible terms of the involved literals and for the moment undetermined coefficients. These polynomials are then added and duplicate terms removed (sums of coefficients are replaced by a single, new coefficient). The terms in these polynomials correspond to the connections in a high order perceptron, and the coefficients to the weights. Hence, only sigmoidal functions have to be added to each polynomial in order to obtain a high order perceptron.

Example: the Boolean terms found by LAD are first separately translated into polynomials:

$$\begin{array}{lcl} b_1 & \Rightarrow & w_1\xi_1 + w_2 \\ \bar{b}_3 & \Rightarrow & w_3\xi_3 + w_4 \\ \bar{b}_1 \wedge b_3 & \Rightarrow & w_5\xi_1\xi_3 + w_5\xi_1 + w_6\xi_3 + w_7 \end{array}$$

Combining these polynomials and interpreting them as connections in a high order perceptron results in (the weights are renumbered):

$$net(\xi) = f(w_1\xi_1\xi_3 + w_2\xi_1 + w_3\xi_3 + w_4)$$

Note that this network contains a second order connection, despite input ξ_2 is unused.

④ Training and ⑤ Pruning

The training of the neural networks is performed using the backpropagation algorithm without further consideration of the fact that the network is constructed in a special way. As the algorithm described above does not give any default values for the weights, they are initialized at random. The networks are pruned using the pruning algorithm found in [Thimm-95] to perform best in terms of final network size for a particular data set, which was either the smallest weight or the smallest connection variance method[4], depending on the data set.

As it turns out that the networks constructed in step ③ are not always capable to learn the task, the maximal order of the network (initially set to 2) and the coverage[5] of the patterns (initially set to 1) are increased one by one until the training in step ④ is successful.

Remark: the networks found by this constructive way have often a different optimal configuration for the training parameters (at least partially caused by the lower number of connections).

7.2 Experiments

The main aim of this section is to show that the generalization capabilities of the constructed networks are comparable to the approach with fully connected *nth*-order perceptrons, but the initial network size is considerably smaller. Each experiment consists of at least 50 simulations. The notation $A \pm x$ in the tables indicates the maximal statistical error of the average A, where X is the maximal difference to the last digit of A (assumed a normal distribution of the data with the average A and a confidence of 95%; see appendix A).

As can be seen in table 7.1, the constructed networks have only between $\frac{1}{16}$ and $\frac{2}{3}$ of the number of connections as compared to the fully connected second order perceptrons, justifying the approach. These networks can be further pruned with success, and the final size of the network is close to what is obtained by pruning the fully connected second order perceptron.

The generalization of the pruned networks is either worse or better as compared to the fully connected second order perceptrons for the same training parameters (this concerns

[4]This method removed connections having the smallest variance for the training set.

[5]Roughly, a *coverage* of n means that for each data pattern with output **true** (**false**) at least n basic patterns, respectively n terms in the Boolean expression, are **true** (**false**); please refer to [Crama-88].

	conv.	$W_{con.}$	W_{2nd}	$W_{con., pr.}$	$W_{2nd, pr.}$	$E_{gen. con.}$	$E_{gen. 2nd}$	Pr. M.
Solar	0.05	26	78	15.7 ± 1	6.8 ± 2	0.106 ± 0	0.081 ± 0	$min(\sigma)$
Auto-mpg	0.06	27	36	6.06 ± 1	8.02 ± 1	0.059 ± 0	0.060 ± 0	$min(w)$
Wine r=4	100%	113	274	22 ± 0	20 ± 1	20.1 ± 1%	17.3 ± 0%	$min(w)$
Wine r=6	100%			19.7 ± 2		16.1 ± 1%		$min(w)$
Servo	0.1	45	91	9.1 ± 1	8.2 ± 1	0.144 ± 0	0.123 ± 0	$min(w)$
Digits	99%	1,260	20,490	292 ± 2	160 ± 2	7.9 ± 0%	9.0 ± 0%	$min(\sigma)$
Glass	0.03	17	121	5.0 ± 0	9.7 ± 0	0.035 ± 0	0.044 ± 0	$min(\sigma)$

conv.	Convergence condition (mean square error, respectively percent recognition if a "%" is given.)
$W_{con.}$ (W_{2nd})	Number of connections in constructed (2nd order) networks
$W_{con., pr.}$ ($W_{2nd, pr.}$)	Average size of constructed (2nd order) and pruned networks
$E_{gen. con.}$ ($E_{gen. 2nd}$)	Generalization error of pruned, constructed (2nd order) networks

Table 7.1: The generalization and size of constructed high order perceptrons.

especially the learning rate). But a different choice can change the outcome of the experiments greatly. For example, experiments with the wine data show that a different choice of the learning rate improves the average generalization of networks produced by the proposed method as compared to a fully connected second order perceptron. Unfortunately, a complete scanning of the domain of learning parameters is hardly feasible due to an high demand of CPU time. It remains therefore unclear whether the networks obtained by applying the proposed method, or the pruning of a fully connected second order neural network give better results for an optimal choice of the learning parameters. A comparison on the base of identical parameters does not permit a conclusive statement, as the different initial size and the presence of connections of an order bigger than two changes the behavior of a high order perceptron and its optimal learning rate. The extreme case that, for the same choice on learning parameters, the constructed high order perceptron converges, but not the fully connected second order perceptron, is imaginable. Therefore the experiments performed show only that the the final network sizes and generalizations are similar, and that the new method is not greatly inferior to the simple approach which prunes fully connected second order networks.

7.3 The Weakness of this Construction Method

As already mentioned, the thresholding of the data set is the weak point in this method. A simple classification problem for which this approach fails is depicted in figure 7.2. The task is to distinguish the *inner* ***I*** and *outer* ***O*** of a square, given the point coordinates x and y. Then, supposed that the points designed in this figure are in the training set, all combinations of Boolean input vectors are mapped onto both classes (the dotted lines in figure 7.2 represent the average of the input elements, respectively the threshold in either directions). Consequently, the presented approach will fail to find a suitable network configuration. It is however unclear, whether this situation is the exception among real world data sets or not. It will be shown latter during the experiments that this approach is successful for at least some data sets.

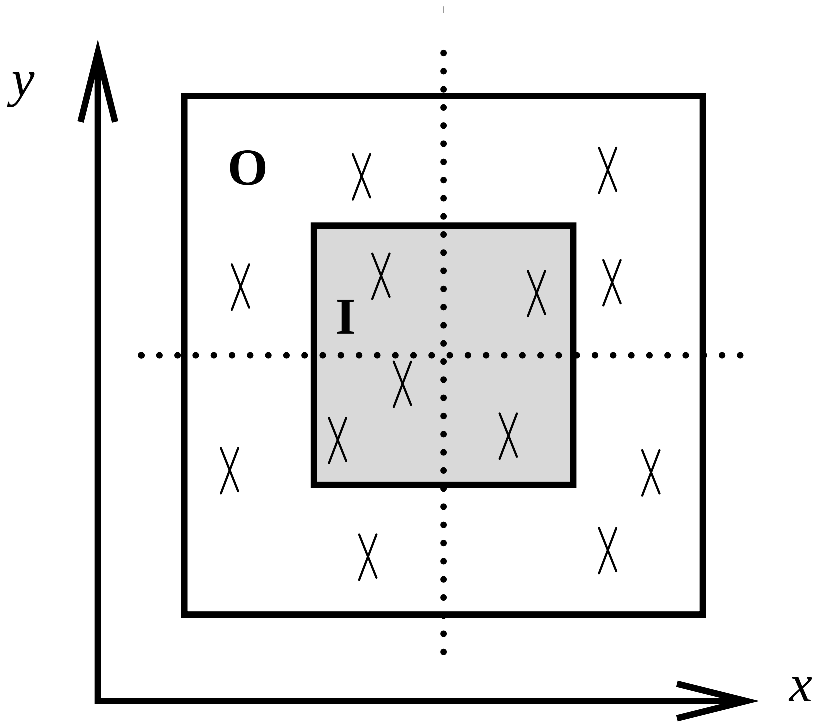


Figure 7.2: A hypothetic data set for which the constructive approach fails.

This method can be developed further: each input data element is not, as described, approximated by one Boolean variable but discretized. Then the resulting data are fed into the algorithm that constructs the Boolean expression. In the next step, all Boolean variables, which emerged from the same input element, are joined together, in order to permit the resulting neural network being applied to the real data set. This can be done by permitting an input value being multiplied with itself. Assume that n discretization levels for a certain input element are necessary to reflect its influence on the output. Then a polynomial of order $\lceil \log_2(n) \rceil$ for this input element is sufficient and could be inserted in the polynomial wherever one of the associated Boolean data elements appears in the basic patterns.

This extension of the approach arises open questions: the most important are the number of the per-input minimal discretization levels and the optimal location for the discretization limits.

7.4 Conclusion

The proposed neural network construction method I proposed reduces significantly the number of connections in the initial network (as compared to the fully connected second order or third order network). It can be expected that this is especially true for data sets with very high dimensional feature vectors. However, the networks are not minimal, and pruning is advantageous. A comparison of pruned fully connected second order with the constructed and pruned networks shows that their sizes as well as their generalizations are comparable.

My idea of using a Boolean approximation of the continuous valued data set to construct an initial network is a generic approach. It can be applied to other architectures and is beneficial if its construction from a Boolean expression has a low complexity. The main requirement of this approach is that an iterative learning rule for the adaption of the weights, such as backpropagation, can be used. Note that other Boolean methods for the construction of an initial neural network may produce better results which is subject to future research.

It can be concluded that the theoretical restriction of my method appears not to be too harsh to prevent a good performance with some real world data sets. However, improvements, which push or eliminate the theoretical limits, are subject to future research.

Chapter 8

Data Analysis and Interpretation of High Order Perceptrons

Statistical approaches for data analysis exist but are either limited to the linear case, like PCA [Harris-75], or computationally very intensive, like non-linear PCA [Oja-93] [Parra-95].

An alternative, although theoretically unfounded approach to data analysis, is to train and to prune a neural network (in the present study high order perceptrons). The resulting neural networks may be analysed, as discussed by R. Andrews *et al.* [Andrews-95]. They also discuss the importance of rule extraction to neural network research and application. Furthermore, it can be regarded as an advantage of the neural networks over (non-linear) PCA and other statistical methods, that it permits different interpretations: the training-pruning approach is likely to produce new networks every run, and therefore different rules or data interpretations will be extracted. Consequently, a developer might chose in a more sensible way among those networks which he estimates being the best (on the base of a comparison of his expert knowledge with the extracted rules *and* the from test data estimated generalization).

However, it is a non-trivial task to develop a formal method for rule extraction and beyond the scope of this dissertation. Therefore, the rule extraction is demonstrated only in an informal and intuitive way in the aim to show its feasibility, as well as to motivate further work in this domain.

8.1 Analysis of the Solar Data Set

This data set contains the sun spot activity for the years 1700 to 1990. The task is to predict the sun spot activity for one of those years, given the activity of the preceding twelve years (see also section 2.4).

A typical result of a training session in chapter 5 is a high order perceptrons described

Connection	Associated Weight	Interpretation
1	1.256	The sun spot activity is likely to increase,
1*2	-1.382	except if the last years the activity was high,
3	-0.659	or it was high 3 years ago.
2*3	1.202	Compensates for 1*2 and 3 (?)
1*8	0.722	If the last year and 8 years ago the activity was hight, then also this year.

Table 8.1: A pruned high order perceptron that was trained on the solar data set.

in table 8.1. Its leftmost column shows the inputs to a connection, where *1* represents the input number of the most recent year (the year before the year to predict). To its right, the corresponding weight and a short informal interpretation is given. Form this table, especially its last line, it can be concluded that the length of the sun spot activity cycle is at least 9 years.

It is clear that the years before the year to predict are most important: four of five connections are connected to the corresponding inputs. Only one connection remained which tracks the long term evolution of the sun spots.

8.2 Miles per Gallon Data set

The data set contains the information described in table 8.2 on cars and are used to predict the city-cycle fuel consumption in miles per gallon gasoline (besides the unused name; see also section 2.4):

input number	interpretation	input number	interpretation
1	cylinders	5	acceleration
2	displacement	6	model year
3	horsepower	7	origin (1 $\leftrightarrow$ America, 2 $\leftrightarrow$ Europe, 3 $\leftrightarrow$ east Asia)
4	weight		

Table 8.2: Data provided in the auto-mpg data set and corresponding inputs.

A network, which resulted from the pruning of a third order perceptron and a short interpretation of the connections, is shown in table 8.3.

8.3 Digit recognition

The training and test data used for this experiment consists of 10x50 handwritten, normalized digits of different writers (compare section 2.4). The initially fully connected second

Connection	Associated Weight	Interpretation
2 3 4	-0.313 -0.281 -0.551	The more displacement, horsepower, or weight, the less miles the car goes with a gallon. Where the weight has the most influence.
6*6	0.530	However, the newer the cars are, the less they consume.
4*6	-0.257	This is even more true for heavy, new cars.
6*6*6	-0.370	This term flattens out the reduction in consumption for the most recent years (note the scaling of the data!).
1*2*4	0.605	However, if the car has a high number of cylinders, displacement and weight, the estimation for the mpg is too low.
6*6*7 6*7*7	0.361 -0.195	New cars from Europe or East Asia consume less than those from America.

Table 8.3: The interpretation of a pruned high order perceptron for the auto-mpg data set.

order perceptron is pruned using the smallest variance method. One of the (smallest for easier analysis in this scope) final neural networks had a performance of 89.6% recognition on untrained data and a total of 129 connections.

Interestingly, the number of connections feeding forward to a certain output differs for the different digits. Table 8.4 shows the number of connections that are connected to the output associated with the detection of a certain digit. The distribution of the connections indicates that the digits *3*, *5*, *8*, and *9* are rather difficult to distinguish for a high order perceptron. Surprisingly, this not true for the digits *1* and *7*.

Digit	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Connections detecting it	7	8	7	13	8	14	10	7	18	13

Table 8.4: Connections per digit in a pruned network.

For illustration, figure 8.1 shows, how the network decides, whether a pattern represents a *0* or not. In this figure, the gray squares represent the pixel of the image, the points the input neurons, and the lines and circles the connections. First order connections are symbolized by circles with one line, second order connections by a circle with two lines connected to it. The color of these circles indicates, whether the activation of a connection favors the detection of a *0* by multiplying its output with a positive value (black) or disfavors it with a negative weight (white). The connections, which are connected to the output indicating that the input represents a *0*, are extracted and drawn in figure 8.1. It can be seen that the right border is not used for the detection of a zero. On the other hand, an activated pixel in the middle of the image means that it unlikely represents a zero.

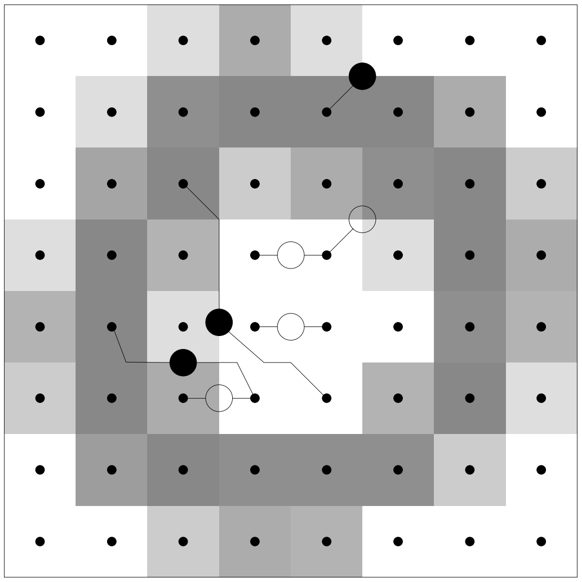


Figure 8.1: The connections deciding upon *0* or not.

Now, one can raise the question which numbers may be misclassified as zero. Consequently, in order to obtain a positive contribution by some connections, a stroke has to cross the pixels in the upper left of the image. On the contrary, the middle of the image has to be blank. This is, for example the case for a *4* which is written in a way that it is normalized to what is displayed in figure 8.2. Then, even if such a *4* is not included in the test data set, the developer can judge the risk that a *4* is misclassified as a *0* too big and reject the network.

8.4 Conclusion

The examples show that pruned high order perceptrons are (at least) informally easy to analyse. It is important to note that both their special architecture and their sparse connectivity simplify the analysis.

The resulting knowledge about how a network recognizes a certain pattern can help to search actively for possibly misclassified patterns which are not among the test data. This can help to increase the confidence in the performance of certain network.

However, an automatic analysis of the data is desirable, which requires a more theoretically founded approach than the one presented in this limited scope.

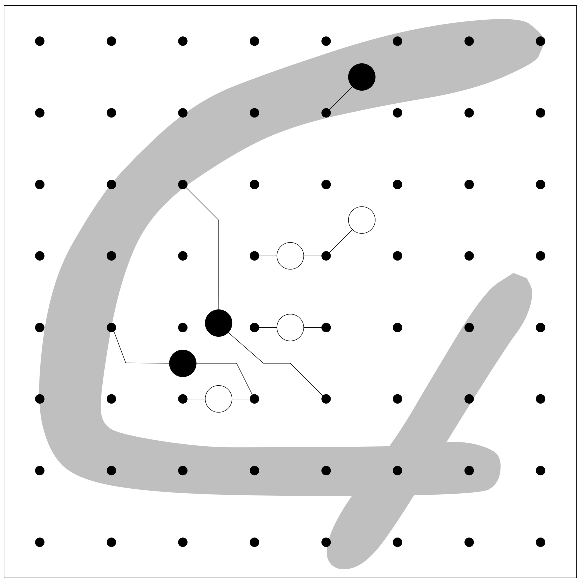


Figure 8.2: A *4* which may be misclassified as a *0*.

Chapter 9

Application of High Order Perceptrons

In this chapter, I validated the efficiency of high order perceptrons and the methods developed in the previous chapters by applying them to various tasks in different domains. Due to the very restricted *a priori* knowledge that is used during the training of a neural network, it is not and can not be the aim to obtain an overall better performance than with task-specialized techniques. It can only be hoped that neural networks perform (at least) almost as good. On the other hand, high order perceptrons and the developed algorithms have the advantage to be easier applicable than task-specialized algorithms: the developer can use neural networks with only little knowledge about the data and still obtain reasonable results in a short time and with little effort.

In the previous chapters, the validation of the performance of high order perceptrons is only done in parts. Although the methods are applied to various data sets, and it is therefore evident that high order perceptrons are applicable, the performance of the resulting networks is not compared with other methods. Anyhow, such a comparison would not be very significant, as the methods are developed using these data and therefore their performance is biased to an unknown extent. Consequently, the data sets used in this chapter have not been used in the research prior to this chapter.

The experiments described in the following sections are performed with two approaches: a fully connected n-th order high order perceptron is pruned and a constructed network is pruned (see chapters 5 and 7). The data sets used during these experiments are split into three parts: a training set, that is used for the training with the backpropagation algorithm, a test set used to chose the network with the best generalization, and a validation set used to estimate the generalization of the latter network.

9.1 Digit Recognition

Data set specification: classification of continuous valued vectors				
Number of patterns in the			Size of the	
training set	test set	validation set	input vector	output vector
10,000	5,000	5,217	64	10

Although the data is extracted from the same database as those used in the previous chapters (see section 2.4), the criticism given in the introduction of this chapter does not fully apply, as a much larger amount of the data set is used.

In this experiment a high order perceptron is constructed using the technique presented in chapter 7. The parameters for the calculation of the Boolean expression are a coverage 2 of the patterns and a maximal order of 3 which resulted in a network with 1953 connections (networks of an order of 2 and a coverage 2 achieved not more than a 75% recognition rate). This network could be trained to a recognition rate of approximately 98.1%. A recognition of 100% could not be reached, which is due to an unknown amount of even for humans unrecognizable characters (which should actually be removed from the training database).

During the training-pruning phase, the test set was used to estimate the generalization of the high order perceptrons. Among the obtained networks, the one with best estimated generalization was selected. This network has 1162 connections and recognized 96.72% of the validation data correctly (only one training session was performed).

For comparison, a multilayer perceptron was trained by Jean-Luc Beuchat on some extracted features. These features, which are known to be efficient for Chinese character recognition, are the locations and heights of the three biggest maxima of four projections (integrals over the gray-level values) of the gray-level images [Cheng-96]. On these 24 values, a multilayer perceptron with 40 hidden neurons (and 1360 connections) was trained. The trained network is capable to recognize 95.62% of untrained data correctly[1].

A recognition rate of 93.5% with a rejection of 2.4% on untrained images of a very similar data base, which is the U.S. Postal Service OAT Handwritten Zip Code Data Base, was reported by S. Knerr *et al.* for a special neural network architecture. Using line segments instead of images, they achieved a recognition rate of 96.5% with a rejection of 1.0% [Knerr-92].

It can be concluded that the application of the construction-pruning approach is successful for this data set: the resulting high order perceptron performs reasonably well, better than a multilayer perceptron trained on some extracted features, and equally good as compared to an approach using line segments and another neural network architecture. High order perceptrons applied to images are even competitive with other types neural networks applied to extracted features.

[1]The reported result is preliminary and part of the diploma work at the EPFL of J.-L. Beuchat on handwritten digit recognition with neural networks.

9.2 Iris Data

Data set specification: classification of continuous valued vectors				
Number of patterns in the			Size of the	
training set	test set	validation set	input vector	output vector
74	37	37	4	3

The task related to this data set is the classification of three iris species, namely the Iris Setosa, the Iris Versicolour, and Iris Virginica. This has to be done by the means of the sepal length and width, as well as the petal length and width (all scaled to the interval $[0, 1]$). Each class (species) is represented by 50 instances. Only one class is linearly separable from the other. The data set was obtained from the UCI data server at CMU [Murphy-96] and was created by R.A. Fisher [Fisher-36].

In a first series of simulations, a fully connected second order network with 42 connections was trained until it recognizes at least all but one pattern. Then the network was pruned following the approach presented in chapter 5, using the smallest variance or the smallest weight method. Then, from each run the smallest network with best performance on the test data set was selected (which was always 100% correct classification). From the obtained networks, the smallest one was chosen and its performance estimated by the means of the validation data set.

This best network contains 9 connections and classifies the entire validation set correctly. For comparison: a standard, first-order perceptron has 12 connections but is unable to perform well on this task as two classes are not linearly separable.

The first run of the constructive approach as described in chapter 7 failed: if a correct classification of all Boolean patterns is required, an almost fully connected second order is constructed, and therefore this method offers no benefit.

In a second run, a 10% misclassification on the Boolean patterns was permitted which resulted in a network with 16 connections. This network was initialized several times with random weights, then trained and pruned as described above. The smallest network found has 10 connections and classifies 100% of the test set correctly. This network recognized 97.3% (all but one pattern) of the validation set correctly.

In conclusion, both basic approaches to search suitable high order perceptrons find networks that are able to separate all three classes almost perfectly. Remarkably, the networks found have less connections than a standard perceptron. For this data set, the algorithms for the training of high order perceptrons yield very satisfactory results.

9.3 Johns Hopkins University Ionosphere Database

Data set specification: classification of continuous valued vectors				
Number of patterns in the			Size of the	
training set	test set	validation set	input vector	output vector
200†	76	75	34	1

† 100 pattern of each class

This database is also obtainable from the UCI data server [Murphy-96] and was used in the original scale (the input elements are in the range of $[-1, 1]$). The following explication comes along with it:

This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts (see [Sigillito-89] for more details). *The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.*

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

In the first experiment, a fully connected second order network (630 connections) was trained to a recognition rate of 99% and pruned with the smallest weight or variance method (all together 20 simulations).

The following was observed:

- The high order perceptron, that performed best on the test set and of a size inferior as compared to a perceptron, recognized 94.7% of the validation set correctly. This network uses only 18 of the 34 input elements and has 48 connections.
- The pruned network, which performed best on the test set (without limitations on its size), recognizes 96% of the validation data correctly and has 229 connections.

The second series of experiments was performed using the constructive method described in chapter 7. The smallest initial network found with this method and learned the training set to 99% recognition, is a third order network, emerging from basic patterns that cover the training set twice. This network has 141 connections and was pruned in 20 simulations. The smallest among the on the test data best performing networks has 34 connections and recognizes 94.7% of the validation data correctly.

The following performances are reported in the documentation of the data set:

- 92% for a perceptron,
- 92.1% for the nearest neighbor algorithm,
- 94.0% for Ross Quinlan's C4 algorithm,
- 96.7% for the IB3 algorithm (see [Aha-89]), and
- 96% on average for a multilayer perceptron.

Comparing these generalizations with those obtained by pruning second order perceptrons shows that high order perceptrons perform as good as multilayer perceptrons, but a little worse than one other method (out of five). The network obtained by the constructive approach has an average performance as compared to the other methods. The performance of both approaches is therefore acceptable for this data set.

9.4 Cadmium Distribution in Lake Geneva

Data set specification: prediction of a continuous valued value from two other				
Number of patterns in the			Size of the	
training set	test set	validation set	input vector	output vector
201	51	59	2	1

This data set describes the cadmium distribution in Lake Geneva by mapping coordinates onto μg cadmium/g water (in the range of $0.08\mu g/g$ to $3.29\mu g/g$). The coordinates as well as the contents of cadmium are scaled to the interval $[0, 1]$. The best result obtained by pruning a fifth order high order perceptron was a network with a mean square error of 0.0166 on the normalized validation set. This figure corresponds to a mean absolute difference of $0.37\mu g/g$. This network contains a fifth and a third order connection.

The application of the constructive approach does not make sense as the data set has only two inputs.

The only other results known for this data set are from personal communications with Perry Moerland at IDIAP and are obtained with multilayer perceptrons. The absolute average generalization error obtained in his experiments is $0.35\mu g/g$ for networks which are pruned to 4 connections.

The best performing network obtained by pruning a fifth order network has a comparable performance to a multilayer perceptron, giving a satisfying result.

9.5 *Ventricular Tachyarrhythmia* Prediction

Data set specification: classification of continuous valued vectors				
Number of patterns in the			Size of the	
training set	test set	validation set	input vector	output vector
37	18	21	13	2

This data set was provided by J.-M. Vesin [Vesin-96]. It contains temporal and spectral indices obtained from series of inter-heartbeat intervals retrieved from implanted defibrillator devices in two conditions:

- just before a *ventricular tachyarrhythmia* (a dangerous cardiac event) and
- in “normal” cardiac conditions.

The task is to predict a *ventricular tachyarrhythmia* in the near future by using the following information:

- the mean length and its standard deviation of inter-heartbeat intervals,
- the mean length and its standard deviation based on 5-minutes intervals,
- the square root of the mean squared differences between successive inter-heartbeat times,
- the percentage of successive inter-heartbeat times whose difference exceeds 50 ms,

- the signal power density VLF_t in the very low frequency band (0 to 0.04 Hz),
- the signal power density LF_t in the low frequency band (0.04 to 0.15 Hz),
- the signal power density HF_t in the high frequency band (0.15 to 0.4 Hz),
- the global signal power density Pt_t in the whole band (0 to 0.4 Hz),
- $LF_t/(Pt_t - VLF_t)$, $HF_t/(Pt_t - VLF_t)$, and LF_t/HF_t.

Preliminary experiments showed that it requires a certain effort to train high order perceptrons to a recognition rate of more than 90% on the training set[2]. Furthermore, the generalization depends much on how the data is distributed on the training, test, and validation sets. The whole data is therefore redistributed on the three sets before every simulation.

The first series of experiments consists of pruning an initially fully connected second order network in 100 simulations with the smallest weight method and in 100 simulations with the smallest variance method. From each convergent simulation, the smallest network with the best generalization on the test set was extracted (the average generalization on the test set is 79% correct classification).

The second series of experiments followed the same scheme, with the exception that the initial network topology is the result of the network construction method described in chapter 7 (using a coverage of 1 and a maximal network order of 2). The result for both series of experiments can be found in table 9.1. Note that in all experiments only about 50% of the initial networks converged within $10,000$ training cycles.

For comparison, an experiment with a standard (first order) perceptron is also documented in table 9.1. This network was trained in 200 simulations on the same, before each simulation re-distributed, data set. A remarkable difference to the experiments with the high order perceptrons is the lower rate of 23% convergent training sessions (again, only these networks are used to estimate the generalization).

	$min(\mathbf{w})$		$min(\sigma)$	
	General.	Size	General.	Size
2nd order perceptrons	$71.4 \pm 2.3\%$	48	$69.9 \pm 1.9\%$	35
Constructed perceptrons	$55.7 \pm 2.4\%$	20	$56.6 \pm 2.3\%$	19

	Unpruned	
	General.	Size
Standard perceptron	$54.5 \pm 2.7\%$	26

The statistical error is calculated for a confidence of 95%.

Table 9.1: Generalization and size of high order perceptrons and standard perceptrons for the *ventricular tachyarrhythmia* data set

Although the results achieved with the pruned, fully connected second order perceptrons are better than those obtained with a simple perceptron, they are unsatisfying. Unfortu-

[2]Even this recognition rate could only be reached with a modification of the on-line learning rule: only misclassified patterns are trained.

nately, no results obtained with other techniques are known by the author, and therefore no founded conclusion is possible. However, the pruned, constructed networks generalize as poorly as a standard perceptron and can be rejected for this application. Note that this is not due to the incapability of the construction method to produce a converging network, as it was criticized in chapter 7, but rather a wrong guess on the right features.

Chapter 10

Conclusion and Outlook

As shown in the previous chapters, I developed high order perceptrons and their training algorithms to a degree which permits their practical application. Furthermore, the number of training parameters specified by the user is reduced: the influence of one of the parameters, namely the gain of the activation function, the distribution of the initial weights, or the learning rate can be neglected. For example, a non-standard gain, which is sometimes imposed by hardware implementations, can be compensated for by changing standard values of the other parameters accordingly. Note that this applies also to many types of feed-forward neural networks and variations of the backpropagation learning rule.

A particular property of high order perceptrons, which permits to eliminate a further parameter, is that, if they are initialized with almost zero initial weights (for example random and with a variance of 10^{-20}), they converge usually in an optimal time, and their performance on untrained patterns is at least as good as for other initial weights. In contrast, the latter does not apply to multilayer perceptrons!

I simplified the choice of the topology: if it is possible to train a fully connected high order perceptron, best results are obtained if these networks are trained and pruned. Two of the examined pruning methods, namely the smallest weight and smallest variance method, produce smaller networks than the others (compare section 5.6). However, no pruning method globally finds networks with better generalization properties, although differences for a particular data set are observable. Furthermore, I showed that not necessarily the smallest network found yields the best generalization.

For some data sets the training of a fully connected second order perceptron is impossible due to a restricted availability of computing time or memory. For these cases I proposed another strategy, called *constructive growing.* This approach uses a Boolean approximation of the data set to estimate which connections are necessary to perform a certain tasks. The construction is then followed by a pruning phase which produces networks of size and generalization comparable to the pruned second order networks.

The fact that the constructive approach does not result in networks that are as good as those produced by the pruning-only approach, leads to the conclusion that the applied constructive method is sub-optimal. This observation together with the theoretical limitations of the algorithm suggest further research in this direction, although it can not be expected that any constructive algorithm will perform better than the pruning-only approach.

Both strategies for the construction of a certain task suitable high order perceptrons are shown to produce networks which perform comparably to other neural networks or statistical methods. Only for one of five data sets the results obtained with the constructive method are poor, and for the same data set the generalization of networks found by the pruning approach are unsatisfying (although no comparison with other methods was possible).

Another general research direction concerning neural networks is rule or knowledge extraction. I therefore informally analysed high order perceptrons trained on some data sets. Although this analysis gives reason to believe that knowledge extraction is formally possible and simpler than for certain other neural network architectures, future research is required.

Appendix A

Confidence

A.1 Confidence Intervals

The confidence intervals are calculated under assumption the distribution of the data in question is normally distributed. In this case, the confidence interval for the real mean μ for n data elements can be calculated on the base of an estimated mean $\bar{x}$ and the estimated standard deviation s (compare page 253ff in [Freund-73]):

$$\bar{x} - z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}} < \mu < \bar{x} + z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}}$$

where $z_{\frac{\alpha}{2}}$ depends on the desired confidence:

Confidence	$z_{\frac{\alpha}{2}}$
0.95	1.96
0.98	2.33
0.99	2.58

A.2 Confidence in Proportions: the 10% and the 90% limit Measure

The confidence in the difference of two sets A and B is judged in the following way: let

$$10\% limit = min(10\%\ limit(A), 10\%\ limit(B)),$$

respectively

$$90\%\ limit = min(90\%\ limit(A), 90\%\ limit(B)).$$

Using these limits, the number of elements x_A and x_B smaller or equal than the 10% respectively 90% limit is determined for each set.

Then the confidence in a set having a smaller 10% limit, respectively 90% limit, is defined as the confidence in the difference between two proportions (compare for example [Freund-73], pp. 317ff). Consequently, the hypothesis *set A is better in the* 10% *limit, respectively* 90% *limit, measure than B* is tested by rejecting the corresponding null hypothesis. The null hypothesis can be rejected with a confidence of 95%, if

$$1.65 \leq \frac{\frac{x_A}{n_A} - \frac{x_B}{n_B}}{\sqrt{p(1-p)(\frac{1}{n_A} + \frac{1}{n_B})}}$$

with $p = \frac{x_A + x_B}{n_A + n_B}$, n_A and n_B the number of elements in set A respectively B, and

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{1.65} e^{\frac{-x^2}{2}} dx \approx 0.95 \quad .$$

Appendix B

Initial Weight Adaptations

B.1 Weight Initialization According to L. F. A. Wessels

L. F. A. Wessels initializes the weights in multilayer perceptrons such that the variance σ_a^2 of the network output is equal to the expected variance σ_t^2 of the target patterns [Wessels-92]. This value is different for (high order) perceptrons as compared to multilayer perceptrons and therefore recalculated for second order networks with linear activation functions. As this value does not depend on the number of outputs, the calculation is done for one output only ($N_1 = d_{in}$ is the number of neurons in the input layer and $\mathcal{E}[x]$ the expectation of x_i for all i):

$$\begin{aligned}
\sigma_a^2 &= \mathcal{E}[a^2] - \mathcal{E}^2[a] = \mathcal{E}[a^2] \quad \text{as weights and input values are independent and the weights are uniform in the interval } [-r, r]. \\
&= \mathcal{E}[\sum_{i,j=1}^{N_1} w_i w_j c_i c_j] \\
&= \sum_{i=1}^{N_1} \mathcal{E}[w_i^2]\mathcal{E}[(c_i)^2] \quad \text{as } \mathcal{E}[w_i w_j] = 0 \text{ if } i \neq j \\
&= \mathcal{E}[w^2] \sum_{i=1}^{N_1} \mathcal{E}[c_i^2] \quad \text{as all weights have the same distribution} \\
&= \mathcal{E}[w^2](\underbrace{N_1 \mathcal{E}[x^2]}_{\text{1st order connections}} + \underbrace{N_1(N_1-1)\mathcal{E}^2[x^2]}_{\text{2nd order with different inputs}} + \underbrace{N_1 \mathcal{E}[x^4]}_{\text{2nd order with same inputs}})
\end{aligned}$$

As the variance σ_w^2 of a variable w is equal to the expectation of its squared value $\mathcal{E}[w^2]$, the equation $\sigma_a^2 = \sigma_t^2$ can be fulfilled by setting

$$\sigma_w^2 = \frac{\sigma_t^2}{N_1\mathcal{E}[x^2] + N_1(N_1-1)\mathcal{E}^2[x^2] + N_1\mathcal{E}[x^4]}$$

For standard perceptrons a similar calculation results in

$$\sigma_w^2 = \frac{\sigma_t^2}{N_1\mathcal{E}[x^2]}$$

$\mathcal{E}[x^2]$ and $\mathcal{E}[x^4]$ can be easily calculated if the distribution of x is known. Using the assumption that x is uniformly distributed and for data scaled to $[-1, 1]$:

$$\mathcal{E}[x^2] = \frac{1}{3} \qquad \mathcal{E}[x^4] = \frac{1}{5}$$

and for the interval $[0, 1]$:

$$\mathcal{E}[x^2] = \frac{1}{12} \qquad \mathcal{E}[x^4] = \frac{1}{80}$$

For the case of a logistic or hyperbolic tangent activation function, an exact calculation is difficult. However, as the output of the network should not be in saturation[1] to perform fast learning, it can be assumed that the weighted sum h_j (see equation 2.5 on page 13) is in the region where the activation function is almost linear.

Consequently, the initial weights should be of a similar variance as for a linear function which is tangent to the actual activation function at the zero point. Thus for the hyperbolic tangent equal to the value calculated above and for the logistic activation function four times as big since $\frac{d}{dx}\left.\frac{1}{1+e^{-x}}\right|_0 = \frac{1}{4}$.

B.2 Weight Initialization According to G. P. Drago *et al.*

G. P. Drago and S. Ridella initialize the weights in the output layer of multilayer perceptrons uniformly in an interval symmetrical to 0. Then the weight vectors are rescaled to have the length $1.3/\sqrt{1+0.3N_1}$. In order to enable a comparison of this approach with others, the resulting initial weights are recalculated in terms of initial weight variances.

Let the r_i be random numbers in the interval $[-b, b]$ (for some arbitrarily chosen b). Then the weights w_i assume the following values:

$$\begin{aligned} w_i &= \frac{r_i}{\sqrt{\sum_{i=1}^{N_1} r_i^2}} \frac{1.3}{\sqrt{1+0.3N_1}} \\ &\approx \frac{r_i}{\sqrt{N_1 \overline{r_i^2}}} \frac{1.3}{\sqrt{1+0.3N_1}} \qquad \text{for big } N_1 \end{aligned}$$

[1] A network is in saturation, when the $|h_i|$ are on average very big, and therefore the output of the networks with a logistic and hyperbolic tangent activation function are always very close to 0 and 1, respectively -1 and 1.

Assumed that the r_i are chosen in the interval $[-b, b]$, the value of $\overline{r_i^2}$ can be approximated by:

$$\frac{1}{2b}\int_{-b}^{b} r^2 dr = \frac{1}{6b}[r^3]_{-b}^{b} = \frac{b^2}{3}$$

Consequently, a random weight initialization in the interval

$$[\frac{-b}{\sqrt{N_1\frac{b^2}{3}}}\frac{1.3}{\sqrt{1+0.3N_1}}, \frac{b}{\sqrt{N_1\frac{b^2}{3}}}\frac{1.3}{\sqrt{1+0.3N_1}}] \approx [\frac{-2.3}{0.55N_1+\sqrt{N_1}}, \frac{2.3}{0.55N_1+\sqrt{N_1}}],$$

which corresponds to the weight variance

$$\sigma_w^2 \approx \frac{1.8}{(0.55N_1+\sqrt{N_1})^2}$$

is comparable to the method proposed by G. P. Drago *et al.* and has the advantage of being computationally less expensive.

B.3 Weight Initialization according to Y. K. Kim *et al.*

The lower bound of the length of a weight vector is calculated by Y. K. Kim and J. B. Ra to be $\sqrt{\eta/d_{in}}$, where η is the learning rate [Kim-91]. This can be recalculated in terms of an interval (compare appendix B.2):

$$[-\sqrt{\frac{3}{N_1}}\sqrt{\frac{\eta}{N_1}}, \sqrt{\frac{3}{N_1}}\sqrt{\frac{\eta}{N_1}}] = [-\frac{\sqrt{3\eta}}{N_1}, \frac{\sqrt{3\eta}}{N_1}],$$

which corresponds to the weight variance

$$\sigma_w^2 \approx \frac{\eta}{(N_1)^2}.$$

B.4 Weight Initialization According to F. J. Śmieja *et al.*

The normalization of the weight vectors to the length $2/\sqrt{d_{in}}$ is similar to an initialization in an interval (compare appendix B.2):

$$[-\sqrt{\frac{3}{N_1}}\frac{2}{\sqrt{N_1}}, \sqrt{\frac{3}{N_1}}\frac{2}{\sqrt{N_1}}] = [-\frac{2\sqrt{3}}{N_1}, \frac{2\sqrt{3}}{N_1}],$$

which corresponds to the weight variance

$$\sigma_w^2 \approx \frac{4}{N_1^2}.$$

Appendix C

The Influence of the Weight Distribution

Tables C.1, C.2, and C.3 show the order ω of the fully interlayer connected network, the activation function f, the convergence criterion or maximal error E, and the learning rate η. The notation '$< a$' means that the mean square distance between network output and target pattern has to be smaller than a, and '$b\%$' means that at least b percent of the patterns must be classified correctly. The columns labelled with the initial weight variances show the outcome of the experiments. A single number corresponds to the mean number of required on-line learning cycles until convergence. A number printed in bold face marks the best result in a row, and an entry p/c signifies that the network did not converge in p percent of the on-line learning cycles, where a trial is judged as non-convergent if the number of cycles exceeds c. The rightmost columns show the maximal radius t_{max} of the confidence intervals[1] for the mean number of required learning cycles for the methods listed in this table and a for random weight initialization with a variance of 0.2.

The networks used in the experiments are of fully connected, and the biases are initialized in the same way as the weights. The only exception is the network trained on the digit data set. This network includes all biases, first order connections, and only those second order connections with both inputs corresponding to different pixels in the same row or the same column in the image (indicated by a "2r" in table C.1). Furthermore, the image size is not as described in the introduction an 8X8 but a 16X16 gray valued image. Training sessions gave an acceptable recognition of untrained digits, despite the small training set used.

Each experiment consists of at least 50 simulations. The number of simulations per experiment was increased until the size of the 95% confidence interval for the mean convergence time permitted a conclusion.

[1]The radius of a confidence interval is the difference between the mean and the upper limit of the interval.

					The initial weight variance σ^2											
	ω	f	E	η	10^{-6}	0.0001	0.001	0.005	0.01	0.05	0.1	0.2	0.5	0.7	1.0	conf.
Solar	1	*id*	<0.08	0.1	**1.3**	1.4	**1.3**	1.4	1.5	1.9	2.5	3.7	6.1	8.6	10.2	1.1, 1.4
	2	*id*	<0.06	0.1	5.6	**5.4**	5.6	6.1	6.6	10.9	15.6	23.4	46.5	58.3	82.0	5.2, 5.7
	3	*id*	<0.06	0.1	**4.5**	**4.5**	4.9	4.9	5.6	9.4	15.4	24.6	53.8	81.9	115.1	4.3, 4.7
	1	f_β	<0.08	0.7	**4.5**	**4.5**	**4.5**	**4.5**	**4.5**	4.7	4.7	4.9	5.3	5.8	6.2	4.4, 4.6
	2	f_β	<0.06	0.7	37.5	**37.1**	37.2	37.4	37.6	38.2	39.0	39.9	45.3	48.4	54.9	36.9, 37.3
	3	f_β	<0.06	0.7	**22.7**	**22.7**	22.8	22.8	22.8	23.5	24.4	26.2	31.5	33.8	40.5	22.5, 22.8
Wine	1	f_β	80%	0.5	242	242	242	242	241	241	241	240	240	237	236	235, 237
	2	f_β	90%	0.5	213	215	214	215	214	214	214	210	**200**	1/500	1/500	197, 204
CES	1	*id*	<0.14	0.1	6.0	6.0	6.2	6.0	6.0	**5.9**	6.4	6.4	8.4	9.3	10.4	5.6, 6.1
	2	*id*	<0.08	0.1	57.6	**57.3**	57.6	58.5	58.0	60.9	63.7	66.0	81.7	81.0	91.2	56.7, 57.9
	1	f_β	<0.14	0.2	12.7	12.7	12.7	12.7	12.6	12.5	12.7	**12.4**	12.7	13.4	14.5	12.0, 12.6
	2	f_β	<0.08	0.2	**158**	159	**158**	**158**	159	159	160	163	167	173	4/300	157, 159
Servo	1	*id*	<0.14	0.01	2.1	**2.0**	2.1	2.7	3.44	5.8	8.0	9.7	11.3	15.6	16.0	1.9, 2.2
	2	*id*	<0.08	0.01	62	**61**	**61**	62	64	71	83	99	159	186	209	61, 62
	1	f_β	<0.14	0.03	19.6	19.6	19.6	19.4	19.2	**18.5**	18.7	19.7	23.7	27.4	35.2	18.1, 19.0
	2	f_β	<0.08	0.03	**168**	**168**	**168**	**168**	**168**	171	172	183	217	231	290	168, 168
Vowels	2	*id*	65%	0.03	**30.6**	31.7	31.5	32.4	32.3	36.3	42.7	51.9	73.2	82.6	103.3	29.8, 31.4
	1	f_β	80%	0.5	64.4	64.5	64.9	64.5	64.6	64.6	63.9	63.8	**63.7**	63.9	64.3	63.3, 64.1
	2	f_β	90%	0.5	15.3	15.3	15.4	15.3	15.3	15.2	15.3	**15.1**	20.0	25.2	4/400	14.9, 15.3
Auto-	2	*id*	<0.06	0.03	10.9	**10.8**	10.9	11.4	11.9	16.3	24.2	32.9	75.3	3/200	24/200	10.7, 11.0
mpg	2	f_β	<0.06	0.1	**22.9**	23.0	**22.9**	23.7	23.2	24.5	26.3	28.9	38.8	45.6	66.8	22.5, 23.3
Glass	1	*id*	<0.04	0.1	4.1	**4.0**	**4.0**	**4.0**	**4.0**	4.4	5.0	5.8	7.6	8.2	9.3	3.8, 4.1
	2	*id*	<0.03	0.005	**13.3**	**13.3**	15.5	30.7	54.6	221	52/400	100/400	100/400	100/400	100/400	13.1, 13.5
Digits	2r	*id*	95%	0.0002	**44.7**	46.8	63.0	119	172	100/100	100/100	100/100	100/100	100/100	100/100	44.1, 45.3

					σ^2		
	ω	f	E	η	2.0	5.0	10.0
Wine	1	f_β	80%	0.5	**234**	4/500	20/500
	2	f_β	90%	0.5	4/500	24/500	58/500

					σ^2	
	ω	f	E	η	10^{-10}	10^{-8}
Digits	2r	*id*	95%	0.0002	45.2	45.2

Table C.1: The performance of high order perceptrons for a uniform weight distribution over the interval $[-a, a]$, with $a = \sqrt{3\sigma^2}$.

	ω	f	E	η	The initial weight variance σ^2											
					10^{-6}	0.0001	0.001	0.005	0.01	0.05	0.1	0.2	0.5	0.7	1.0	conf.
Solar	1	*id*	<0.08	0.1	**1.3**	**1.3**	1.5	**1.3**	1.5	1.5	1.6	2.5	5.6	7.4	10.2	1.2, 1.4
	2	*id*	<0.06	0.1	5.5	**5.4**	5.5	5.5	5.5	6.3	7.9	14.8	39.0	55.4	79.8	5.3, 5.5
	3	*id*	<0.06	0.1	**4.5**	**4.5**	**4.5**	4.8	4.8	5.6	8.4	14.4	45.1	68.4	111.1	4.4, 4.7
	1	f_β	<0.08	0.7	4.5	4.5	4.5	4.5	**4.4**	4.5	4.6	4.7	5.2	5.8	6.4	4.4, 4.5
	2	f_β	<0.06	0.7	37.3	37.4	37.4	**37.2**	**37.2**	37.3	37.7	38.9	43.9	48.1	54.6	37.3, 37.6
	3	f_β	<0.06	0.7	**22.6**	**22.6**	22.7	22.7	22.7	22.9	23.0	24.7	29.6	34.2	38.8	22.5, 22.8
Wine	1	f_β	80%	0.5	242	242	242	242	242	242	242	241	238	240	236	227, 239
	2	f_β	90%	0.5	215	215	215	213	214	213	216	215	204	203	**199**	196, 202
CES	1	*id*	<0.14	0.1	**6.0**	**6.0**	**6.0**	**6.0**	6.1	**6.0**	**6.0**	6.4	7.5	8.7	9.7	5.9, 6.1
	2	*id*	<0.08	0.1	57.7	57.8	57.7	57.8	**57.6**	58.1	59.5	64.2	74.4	81.6	98.7	57.4, 57.8
	1	f_β	<0.14	0.2	12.4	12.9	12.5	12.7	12.8	12.8	12.5	12.5	**12.1**	13.4	14.7	11.4, 12.8
	2	f_β	<0.08	0.2	158	158	158	158	158	**157**	160	160	167	2/300	6/300	155, 158
servo	1	*id*	<0.14	0.01	2.1	**1.9**	2.0	2.1	2.1	2.8	4.7	7.7	12.7	13.3	14.7	1.8, 2.0
	2	*id*	<0.08	0.01	**62**	**62**	**62**	**62**	**62**	63	66	81	145	164	231	61, 62
	1	f_β	<0.14	0.03	19.6	19.6	19.7	19.6	19.6	**19.3**	**19.3**	19.5	22.5	25.6	34.2	19.1, 19.4
	2	f_β	<0.08	0.03	168	168	168	168	168	**167**	170	173	211	237	2/500	166, 168
Vowels	2	*id*	65%	0.03	31.7	32.4	32.1	**30.0**	31.5	32.4	34.2	41.7	68.1	80.9	100.8	28.9, 31.1
	1	f_β	80%	0.5	64.4	64.6	64.5	64.7	65.0	64.4	64.9	64.0	63.7	**63.3**	63.7	62.6, 64.0
	2	f_β	90%	0.5	15.7	15.4	15.5	15.3	15.4	15.5	**15.2**	15.4	16.2	24.7	4/400	15.0, 15.4
Auto-	2	*id*	<0.06	0.03	10.8	10.9	10.8	**10.7**	**10.7**	11.3	13.6	23.0	69.1	3/200	28/200	10.6, 10.9
mpg	2	f_β	<0.06	0.1	22.8	**22.7**	22.9	22.9	22.8	23.1	24.0	26.6	37.0	45.0	67.3	22.5, 22.9
Glass	1	*id*	<0.04	0.1	4.2	4.1	**4.0**	4.1	**4.0**	4.3	4.2	4.9	6.9	8.1	9.1	3.8, 4.1
	2	*id*	<0.03	0.005	**13.3**	**13.3**	**13.3**	13.4	13.8	39.4	128	48/400	100/400	100/400	100/400	13.2, 13.4

	ω	f	E	η	σ^2		
					2.0	5.0	10.0
Wine	1	f_β	80%	0.5	**234**	1/500	16/500
	2	f_β	90%	0.5	4/500	16/500	36/500

Table C.2: The performance of high order perceptrons for a normal distribution restricted to the interval $[-3\sigma^2, 3\sigma^2]$.

	ω	f	E	η	σ^2 10^{-6}	0.0001	0.001	0.005	0.01	0.05	0.1	0.2	0.5	0.7	1.0	conf.
Solar	1	*id*	<0.08	0.1	**1.2**	**1.2**	1.4	1.4	1.4	2.1	2.7	3.5	6.5	9.3	11.5	1.1, 1.3
	2	*id*	<0.06	0.1	5.4	**5.3**	5.6	5.8	6.7	10.1	15.0	24.1	46.0	60.2	83.5	5.2, 5.5
	3	*id*	<0.06	0.1	**4.5**	4.6	4.7	5.2	5.8	10.2	14.7	24.4	56.2	77.8	114	4.3, 4.6
	1	f_β	<0.08	0.7	4.5	**4.4**	4.5	4.5	4.6	4.8	4.7	4.8	5.4	5.5	6.5	4.3, 4.5
	2	f_β	<0.06	0.7	37.4	**37.1**	**37.1**	37.4	37.3	37.9	39.1	41.0	47.9	48.6	53.7	36.9, 37.4
	3	f_β	<0.06	0.7	22.7	**22.6**	22.8	**22.6**	23.0	23.5	24.4	26.3	31.3	34.8	38.9	22.5, 22.8
Wine	1	f_β	80%	0.5	242	242	242	242	242	241	241	241	240	238	237	231, 234
	2	f_β	90%	0.5	215	215	215	214	214	214	214	210	208	**198**	**198**	194, 202
CES	1	*id*	<0.14	0.1	6.0	6.0	**5.9**	6.1	6.0	6.0	6.0	6.6	9.0	9.8	10.5	5.7, 6.0
	2	*id*	<0.08	0.1	**56.7**	58.0	57.7	57.8	58.5	59.8	65.8	70.5	85.1	84.1	94.7	56.1, 57.3
	1	f_β	<0.14	0.2	12.6	12.7	12.8	12.5	12.5	12.6	**12.3**	12.5	12.8	12.8	14.5	11.8, 12.7
	2	f_β	<0.08	0.2	159	**158**	**158**	**158**	159	160	162	164	176	173	7/300	157, 159
Servo	1	*id*	<0.14	0.01	**2.1**	2.3	**2.1**	2.8	3.0	5.9	7.7	10.0	12.7	13.5	17.3	1.9, 2.3
	2	*id*	<0.08	0.01	61.4	61.6	**61.1**	62.6	64.9	69.3	83.5	100	153	195	225	60.4, 61.8
	1	f_β	<0.14	0.03	19.6	19.6	19.5	19.6	19.5	**19.1**	19.2	19.7	23.4	27.7	33.7	18.7, 19.4
	2	f_β	<0.08	0.03	168	168	**167**	168	168	172	176	182	221	253	309	167, 168
Vowels	2	*id*	65%	0.03	30.9	32.9	**30.8**	32.9	33.4	38.4	41.1	50.8	73.0	85.9	101.2	29.5, 32.2
	1	f_β	80%	0.5	64.3	64.9	64.4	63.6	64.2	63.9	63.7	64.0	63.4	**63.2**	63.7	62.5, 63.9
	2	f_β	90%	0.5	15.3	15.3	15.2	15.4	15.6	**14.9**	15.5	15.6	17.9	21.9	5/400	14.6, 15.1
Auto-mpg	2	*id*	<0.06	0.03	**10.8**	**10.8**	10.9	11.5	11.8	17.0	22.5	34.0	74.0	10/200	28/200	10.5, 11.1
	2	f_β	<0.06	0.1	23.0	22.8	**22.7**	23.4	23.5	24.7	26.4	28.2	43.5	49.1	69.8	22.3, 23.1
Glass	1	*id*	<0.04	0.1	4.0	**3.9**	4.1	4.2	4.1	4.5	5.0	6.0	7.2	8.7	9.6	3.7, 4.1
	2	*id*	<0.03	0.005	**13.4**	**13.4**	15.3	29.5	52.2	223	56/400	100/400	100/400	100/400	100/400	13.2, 13.6

	ω	f	E	η	σ^2 2.0	5.0	10.0
Wine	1	f_β	80%	0.5	**233**	4/500	32/500
	2	f_β	90%	0.5	4/500	12/500	34/500

Table C.3: The performance of high order perceptrons for a uniform weight distribution over the intervals $[-2a, -a]$ and $[a, 2a]$, with $a = \sqrt{3\sigma^2/7}$.

Appendix D

Implementation of a Simulator

This chapter is a short summary of important properties that neural network simulator should possess and reflects the experience gained by using several simulators (compare [Thimm-94.1]): *Sesame*[1] [Linden-94] [Linden-92], *OpenSimulator*[2] [Leber-92] [Leber-93], and *SNNS* [Zell-95]. Many other exist, a few of them are: the *MLC++ utilities* [Kohavi-95], the *PDP++ simulator* [O'Reilly-95], and *GENESIS* [Bower-95.1] [Bower-95.2].

In the aim to help a prospective neural network researcher to chose a simulator, only (dis-)advantages and encountered problems are summarized. A detailed discussion of the simulators is not very useful, as *Sesame* and *OpenSimulator* are apparently not anymore supported, and *SNNS* has serious problems with bugs (in the opinion of the author out-ranging it for professional use);

***Ontogenic* features** [3] Many neural network architectures proposed in the last years possess ontogenic features. The structure of many simulators does not always support the implementation of these features. Changing the simulator of choice towards being ontogenic, results either in kludges or a considerable amount of programming.

High order and other types of non-linear connections None of the simulators used contains conceptually high order or other types of non-linear connections. A corresponding extension of the simulator may be difficult if the data structures of the simulator are not flexible enough.

Graphic capabilities are important, as research on neural network often requires a detailed observation of some values.

[1] *Sesame* is copyrighted by the GMD Schloß Birlinghoven.

[2] *OpenSimulator* is copyrighted by the ETH Zürich.

[3] The term *ontogenic* means that the topology of is modified in some way, usually by adding or removing units.

Batch processing In current neural network research, simulation plays a crucial role. It is often important to run a high amount of simulations which is preferably done in a "batch" and therefore without any graphics. Two versions to achieve this been found: either all graphic commands are encapsulated in a conditional `if graphics then...`, or two versions of a simulator exist: one with, the other without graphics. The latter appeared to require more programming efforts, as two programs have to be maintained.

Integration into delivered software In applications, neural networks have to be integrated and combined with other modules. It is therefore important that the neural network found during a training session can be extracted from the simulation tool and integrated into the software delivered to the users. Some applications might even require training during the normal usage, implying that (parts of) the simulator have to be incorporated into a program.

Simulations It is important for a fast and easy performance of experiments that various network training parameters can be stored and changed easily by a graphical interface or a command interpreter without re-starting the simulator. It is a further advantage if this also applies to the setup of the simulator configuration (location and size of windows *etc.*).

Neural network topologies Although a wide range of neural network simulators is available, it is impossible to foresee or even to keep up with the continuous surge of new neural networks and their variations. Only a few simulators are flexible enough to facilitate substantial topology alterations (see for example [Fiesler-94] and [Fiesler-94] for information on neural network topologies). An on-line combination of modules is a big advantage.

Implementation The most preferred implementation languages are *C*++, C, but more general tools like Matlab [4] provide modules for neural network simulation. Modular object-oriented neural network simulators promise that modifications and extensions to be made with minimal effort. Object-oriented programming languages facilitate the fulfillment of these demands for a flexible simulator, especially if many reusable modules for neural network models and data handling (vector and matrix operations, statistical analysis, pattern and file handling, and graphical displays) exist. An important subject is also the ease with which the simulator can be maintained; changes due to a modification of a neural network should be localized to a (few) modules.

Parallelism The distribution of simulations over a computer network may or may not be required.

Documentation Especially for the implementation of a new neural network architecture it is important that the sources are well-documented.

[4] Matlab is under copyright ©by The MathWorks, Inc., 1994

Bibliography

[Aha-89] David W. Aha and Dennis Kibler. "Noise-Tolerant Instance-Based Learning Algorithms." In N. S. Sridharan (editor), *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, (ISBN: 1-55860-094-9), volume 1, pages 794–799. Morgan Kaufmann, Detroit, MI, USA, August 1989.

[Andrews-95] Robert Andrews, Joachim Diderich, and Alan B. Tickle. "A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks." Technical report, Queensland University of Technology, Brisbane Q 4001, Australia, 1995.

[Bachmann-90] Charles McKay Bachmann. "*Learning and Generalization in Neural Networks.*" PhD thesis, Department of Physics, Brown University, Providence, Rhode Island, USA, May 1990.

[Bartlett-94] Eric B. Bartlett. "Dynamic Node Architecture Learning: an Information Theoretic Approach." *Neural Networks*, volume 7, number 1, pages 129–140, 1994.

[Beiu-95] Valeriu Beiu and John G. Taylor. "Area-Efficient Constructive Learning Algorithm." In *Proceedings of CSCS-10: the 10th International Conference on Control System and Computer Science*, volume 5, pages 293–310. Bucharest, Rumania, 1995.

[Bellido-93] I. Bellido and E. Fiesler. "Do Backpropagation Trained Neural Networks have Normal Weight Distributions?" In Stan Gielen and Bert Kappen (editors), *ICANN '93; Proceedings of the International Conference on Artificial Neural Networks*, (Amsterdam, The Netherlands; 13–16 September, 1993), (ISBN: 0-387-19839-3), pages 772–775. Springer-Verlag, London, UK, 1993.

[Beuchat-95] Jean-Luc Beuchat. "Optimisation de réseaux de neurones." Report of a student project performed at IDIAP, C.P. 592, 1920 Martigny, Switzerland, 1995.

[Bodenhausen-91] Ulrich Bodenhausen and Alex Waibel. "The Tempo 2 Algorithm: Adjusting Time-Delays by Supervised Learning." *Advances in Neural Informationn Processing Systems*, volume 3, 1991.

[Boers-92] Egbert J. W. Boers and Herman Kuiper. "Biological Metaphors and the Design of Modular Artificial Neural Networks." Master's thesis, Leiden University, Leiden, The Netherlands, August 1992.

[Boros-96] Endre Boros, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, Eddy Mayoraz, and Ilya Muchnik. "An Implementation of Logical Analysis of Data." IDIAP-RR 5, IDIAP, CP 592, 1920 Martigny, Switzerland, 1996.

[Bottou-88] Léon-Yves Bottou. "Reconnaissance de la parole par réseaux multi-couches." In *Neuro-Nîmes'88; Proceedings of the International Workshop on Neural Networks and Their Applications*, (EC2 and Chambre de Commerce et d'Industrie de Nîmes; Nîmes, France; November 15–17, 1988), (ISBN: 2-906899-14-3), pages 197–217. 269–287, rue de la Garenne, 92000 Nanterre, France, 1988.

[Bower-95.1] James M. Bower and David Beeman. "*The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.*" TELOS/Springer-Verlag, 1995.

[Bower-95.2] James M. Bower and David Beeman. "GENESIS 2.0." Obtainable via anonymous ftp from genesis.bbb.caltech.edu the file /pub/genesis and the URL http://www.bbb.caltech.edu/GENESIS. See also [Bower-95.2].

[Burgess-92] Neil Burgess, Silvano Di Zenzo, Paolo Ferragina, and Mario Notturno Granieri. "The Generalization of a Constructive Algorithm in Pattern Classification Problems." *International Journal of Neural Systems*, supplement to volume 3, 1992.

[Campell-95] Colin Campbell and C. Perez Vicente. "The Target Switching Algorithm: a Constructive Learning Procedure for Feed-Forward Neural Networks." *Neural Computation*, volume 7, number 6, pages 1245–1264, November 1995.

[Chen-91] C.L. Chen and R.S. Nutter. "Improving the Training Speed of Three-Layer Feedforward Neural Nets by Optimal Estimation of the Initial Weights." In *International Joint Conference on Neural Networks*, volume 3, pages 2063–2068. IEEE, 1991.

[Cheng-96] H. D. Cheng and C. D. Xia. "A novel parallel approach to character recognition and its VLSI implementation." *Pattern Recognition*, volume 29, number 1, pages 97–119, January 1996.

[Cho-91] T.-H. Cho, R.W. Conners, and P.A. Araman. "Fast Backpropagation Learning Using Steep Activation Functions and Automatic Weight Reinitialization." In *Conference Proceedings 1991 IEEE International Conference on Systems, Man, and Cybernetics. 'Decision Aiding for Complex Systems'(Cat. No.91CH3067-6)*, (Charlottesville, VA, USA; October 13–16, 1991), (ISBN: 0 7803 0233 8), volume 3, pages 1587–92. IEEE, New York, NY, USA, 1991.

[Choi-96] Ju-Yeop Choi, Hugh F. VanLandingham, and Sanoje Bingulac. "A Constructive Approach for Nonlinear System Identification Using Multilayer Perceptrons." *IEEE Transactions on Systems, Man, and Cybernetics*, volume 6, number 2, pages 307–312, April 1996.

[Corwin-94] Edward M. Corwin, Antonette M. Logar, and William J. B. Oldham. "An Iterative Method for Training Multilayer Networks with Threshold Functions." *IEEE Transactions on Neural Networks*, volume 5, number 3, pages 507–508, May 1994.

[Crama-88] Yves Crama, Peter L. Hammer, and Toshhihide Ibaraki. "Cause-effect Relationships and Partially Defined Boolean Functions." Rutcor Research Report # 39-88, Rutcor, Rutgers University, New Brunswick, NJ 08903, USA, August 1988.

[Darken-92] Christian Darken, Joseph Chang, and John Moody. "Learning Rate Schedules for Faster Stochastic Gradient Search." In S. Y. Kung, F. Fallside, J. Aa. Sorenson, and C. A. Kamm (editors), *Neural Networks for Signal Processing II; Proceedings of the 1992 IEEE-SP Workshop*, (IEEE-SP; Helsingoer, Denmark; August 31 – September 2, 1992), (ISBN: 0-7803-0557-4), pages 1–12. Piscataway, New York, USA, 1992.

[Denoeux-93] Thierry Denoeux and Régis Lengellé. "Initializing Back Propagation Networks with Prototypes." *Neural Networks*, volume 6, pages 351–363, 1993.

[Deterding-89] David Deterding, Mahesan Niranjan, and Tony Robinson. "British Vowels Data Set." via anonymous ftp from the UCI Repository (see **[data-sever]**), 1989. This data set is maintaned by the CMU; email: neural-bench@cs.cmu.edu.

[Drago-92] Gian Paolo Drago and Sandro Ridella. "Statistically Controlled Activation Weight Initialization (SCAWI)." *IEEE Transactions on Neural Networks*, volume 3, number 4, pages 627–631, July 1992.

[Efron-95] Bradley Efron and Robert Tibshirani. "Cross-Validation and the Bootstrap: Estimating the Error Rate of a Prediction Rule." Technical Report. Obtainable via anonymous ftp from utstat.toronto.edu:pub/bootpred.shar or ordered from karola@playfair.stanford.edu, 1995.

[Ekeberg-89] Örjan Ekeberg and Anders Lansner. "Automatic Generation of Internal Representations in a Probabilistic Artificial Neural Network." In L. Personnaz and G. Dreyfus (editors), *Neural Networks from Models to Application; Proceedings of nEuro-88, The First European Conference on Neural Networks*, (E.S.P.C.I.; Paris, France), (ISBN: 2-903667-03-9), pages 178–186. I.D.S.E.T., Paris, 10 rue Vauquelin, 75005 Paris, France, 1989.

[Fahlman-88] Scott E. Fahlman. "An Empirical Study of Learning Speed in Back-Propagation Networks." Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, September 1988.

[Fahlman-90] Scott E. Fahlman and Christian Lebiere. "The Cascade-Correlation Learning Architecture." In David S. Touretzky (editor), *Advances in Neural Information Processing Systems (NIPS)*, (IEEE; Denver, Colorado; November 27–30, 1989), (ISBN: 1-55860-100-7), volume 2, pages 524–532. Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, California 94403, 1990.

[Fahner-92] Gerald Fahner, Nils Goerke, and Rolf Eckmiller. "Structural Adaptation of Boolean Higher Order Neurons: Classification with Parsimonious Topologies for Superior Generalization." In Igor Alexander and John Taylor (editors), *Artificial Neural Networks, 2: Proceedings of the 1992 International Conference on Artificial Neural Networks (ICANN-92)*, (Brighton, United Kingdom; 4–7 September 1992), (ISBN: 0-444-89488-8), volume 1, pages 285–288. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[Fahner-94] Gerald Fahner and Rolf Eckmiller. "Structural Adaptation of Parsimonious Higher-Order Neural Classifiers." *Neural Networks*, volume 7, number 2, pages 279–289, March 1994.

[Fiesler-93] E. Fiesler. "Minimal and High Order Neural Network Topologies." In *Proceedings of the Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defense; An International Conference on Computational Intelligence: Neural Networks, Fuzzy Systems, Evolutionary Programming and Virtual Reality (WNN93/FNN93)*, (San Francisco, California; 7–10 November, 1993), (ISBN: 1-56555-059-5), number 2204 in SPIE Proceedings, pages 173–178. Simulation Councils, Inc. / The Society for Computer Simulation, San Diego, California, 1993.

[Fiesler-94] E. Fiesler. "Comparative Bibliography of Ontogenic Neural Networks." In Maria Marinaro and Pietro G. Morasso (editors), *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*, (Sorrento, Italy; 26–29 May, 1994), (ISBN: 3-540-19887-3), volume 1, pages 793–796. Springer-Verlag, London, UK, 1994.

[Fiesler-94] E. Fiesler. "Neural Network Classification and Formalization." *Computer Standards & Interfaces*, volume 16, number 3, pages 231–239, June 1994.

[Fiesler-96] E. Fiesler and K. Cios. "Supervised Ontogenic Networks." In E. Fiesler and R. Beale (editors), *The Handbook of Neural Computation*, (ISBN 0-7503-0312-3), part C1.7. Oxford University Press and IOP Publishing, 198 Madison Avenue, New York, NY 10016, 1996.

[Fiesler-97] E. Fiesler and R. Beale (editors). "*Handbook of Neural Computation*," (ISBN 0-7503-0312-3), Oxford University Press and IOP Publishing, 198 Madison Avenue, New York, NY 10016, 1997.

[Finnoff-93] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. "Improving Model Selection by Nonconvergent Methods." *Neural Networks*, volume 6, pages 771–783, 1993.

[Fisher-36] R.A. Fisher. "The use of multiple measurements in taxonomic problems." *Annual Eugenics*, volume 7, number II, pages 179–188, 1936.

[Freund-73] John. E. Freund. "*Modern Elementary Statistics*." Prentice–Hall, Engelwood Cliffs, New Jersey, forth edition, 1973.

[Friedman-96] Jerome H. Friedman. "Local Learning Based on Recursive Covering." Technical report, Department of Statistics, Stanford Linear Accelerator Center, Stanford University, Email: jhf@playfair.stanford.edu, 1996. Available via anonymous ftp from playfair.stanford.edu the file /pub/friedman/dart.ps.Z.

[Garris-92] M. D. Garris and R. A. Wilkinson. "*NIST Special Database 3*." National Institute of Standarts and Technology, Advanced System Division, Image Recognition Group, February 1992.

[Ghosh-92] Joydeep Ghosh and Yoan Shin. "Efficient Higher-Order Neural Networks for Classification and Function Approximation." *International Journal of Neural Systems*, volume 3, number 4, pages 323–350, 1992.

[Giles-95] C. Lee Giles, Dong Chen, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chung Lee, and Mark W. Goudreau. "Construnctive Learning of Recurrent Neural Networks: Limitations of Recurrent Cascade Correlation and a Simple Solution." *IEEE Transactions on Neural Networks*, volume 6, number 4, pages 829–836, July 1995.

[Girosi-95] Frederico Girosi, Michael Jones, and Tomaso Poggio. "Regularization Theory and Neural Networks Architectures." *Neural Computation*, volume 7, number 2, pages 219–269, March 1995.

[Gorodkin-93] J. Gorodkin, L.K. Hansen, A. Krogh, C. Svarer, and O. Winther. "A Quantitative Study of Pruning by Optimal Brain Damage." *International Journal of Neural Systems*, volume 4, number 2, pages 159–69, June 1993.

[Gosh-94] Joydeep Ghosh and Kagan Tumer. "Structural Adaptation and Generalization in Supervised Feed-Forward Networks." *Journal of Artificial Neural Networks*, volume 1, number 4, pages 431–458, 1994.

[Gray-92] Donald L. Gray and Anthony N. Michel. "A Training Algorithm for Binary Feedforward Neural Networks." *IEEE Transactions on Neural Networks*, volume 3, number 2, pages 176–194, March 1992.

[Haffner-88] P. Haffner, A. Waibel, H. Sawai, and K. Shikano. "Fast Back-Propagation Learning Methods for Neural Networks in Speech." Technical Report TR-1-0058, ATR Interpreting Telephony Research Laboratories, Osaka, Japan, 1988.

[Hanson-89] Stephen José Hanson and Lorien Y. Pratt. "Comparing Biases for Minimal Network Construction with Back-Propagation." In David S. Touretzky (editor), *Advances in Neural Information Processing Systems 1*, (IEEE; Denver, Colorado; 1988), (ISBN: 1-558-60015-9), pages 177–185. Morgan Kaufmann, San Mateo, California, 1989.

[Harris-75] Richard J. Harris. "*A Primer of Multivariate Statistics.*" Academic Press, Inc., 1975.

[Hassibi-92] Babak Hassibi and David G. Stork. "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon." Technical Report CRC-TR-9214, RICOH California Research Center, Menlo Park, California, USA, May 7, 1992.

[Hertz-91] J. Hertz, A. Krogh, and R. G. Palmer. "*Introduction to the Theory of Neural Computation.*" volume I of *Computation and Neural Systems Series; Santa Fe Institute Studies in the Sciences of Complexity; Lecture notes.* Addison-Wesley Publishing Company, The Advanced Book Program, Redwood City, California, 1991.

[Heywood-93] Malcolm I. Heywood and Peter D. Noakes. "Simple Addition to Back-Propagation Learning for Dynamic Weight Pruning, Sparse Network Extraction and Faster Learning." In *Proceedings of the 1993 IEEE International Conference on Neural Networks*, (IEEE; San Francisco, California; March 28 – April 1, 1993), (ISBN: 0-7803-1200-7), volume II, pages 620–625. IEEE, Piscataway, New Jersey, 1993.

[Heywood-94] Malcolm I. Heywood and Peter D. Noakes. "Directed Product Term Selection in a Sigma - Pi Networks." In *Proceedings of the IEEE International Conference on Neural Networks (ICNN94)*, (IEEE; Orlando, Florida, USA; June 27–29, 1994), (ISBN: 0-7803-1901-x), volume 1, pages 489–493. IEEE Service Center, Piscataway, NJ, USA, 1994.

[Higgins-91] C.M. Higgins and R.M. Goodman. "Incremental learning with rule-based neural networks." In *IJCNN-91-Seattle: International Joint Conference on Neural Networks (Cat. No.91CH3049-4)*, (IEEE Int. Neural Network Soc; Seattle, WA, USA; 8-14 July 1991), (ISBN: 0 7803 0164 1), volume 1, pages 875–80. IEEE, New York, NY, USA, 1991.

[Hochreiter-94] Sepp Hochreiter and Juergen Schmidhuber. "Flat Minimum Search Finds Simple Nets." Technical Report FKI-200-94, Fakultät für Informatik, Technische Universität München, 80290 München, Germany, December 1994.

[Hochreiter-96] Sepp Hochreiter and Jürgen Schmidhuber. "Flat Minima." *Neural Computation.* Accepted for publication, 1996.

[Hush-92] Don R. Hush, Bill Horne, and John M. Salas. "Error surfaces for Multilayer Perceptrons." *IEEE Transactions on Systems, Man and Cybernetics*, volume 22, number 5, pages 1152–1161, September/October 1992.

[Izui-90] Yoshio Izui and Alex Pentland. "Speeding Up Back Propagation." In Maureen Caudill (editor), *Proceedings of the International Joint Conference on Neural Networks (IJCNN-90-WASH-DC)*, (INNS and IEEE; Washington, DC; January 15–19, 1990), (ISBN: 0-8058-0754-3), volume I; Theory Track, Neural and Cognitive Sciences Track, pages 639–642. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1990.

[Jackson-96] Jeffrey C. Jackson and Mark W. Craven. "Learning Sparse Perceptrons." In *Advances in Neural Information Processing Systems*, volume 8. MIT Press, Cambridge, MA, 1996.

[Jia-94] Q. Jia, K. Hagiwara, N. Toda, and S. Usu. "Equivalence Relation Between the Backpropagation Learning Process of an FNN and That of an FNNG." *Neural Networks*, volume 7, number 2, page 411, 1994.

[Judge-85] George G. Judge, W. E. Griffiths, R. Carter Hill, and Tsoung-Chao Lee. "*The Theory and Practice of Econometrics.*" Wiley Series in Probability and mathematical statistics. John Wiley and Sons, 2 edition, 1985.

[Karnin-90] E. D. Karnin. "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks." *IEEE Transactions on Neural Networks*, volume 1, number 2, pages 239–242, June 1990.

[Kim-91] Y. K. Kim and J. B. Ra. "Weight Value Initialization for Improving Training Speed in the Back Propagation Network." In *International Joint Conference on Neural Networks*, volume 3, pages 2396–2401. IEEE, 1991.

[Knerr-92] Stefan Knerr, Léon Personnaz, and Gérard Dreyfus. "Handwritten Digit Recognition by Neural Networks with Single-Layer Training." *IEEE Transactions on Neural Networks*, volume 3, number 6, pages 962–968, November 1992.

[Kohavi-95] Ronny Kohavi. "MLC++ Utilities." The package can be obtained using the URL http://www.sgi.com/Technology/mlc/.

[Kolen-90] John F. Kolen and Jordan B. Pollack. "Back Propagation is Sensitive to Initial Conditions." Technical Report TR 90-JK-BPSIC, Laboratory for Artifical Intelligence Research, Computer and Information Science Department, The Ohio State University, Columbus, Ohio 43210, USA, kolen-j@cis.ohio-state.edu, 1990.

[Kowalczyk-94] Adam Kowalczyk and Herman L. Ferrá. "Developing Higher-Order Networks With Empirically Selected Units." *IEEE Transactions on Neural Networks*, volume 5, number 5, pages 698–711, September 1994.

[Kruschke-91] John K. Kruschke and Javier R. Movellan. "Benefits of Gain: Speeded Learning and Minimal Hidden Layers in Back-Propagation Networks." *IEEE Transactions on Systems, Man and Cybernetics*, volume 21, number 1, pages 273–280, January/February 1991.

[Kwok-95] Tin-Yau Kwok and Dit-Yan Yeung. "Constructive Feedforward Neural Networks for Regression Problems: a Survey." Technical Report HKUST-CS95-43, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, jamesk,dyyeung@cs.ust.hk, 1995.

[Le Cun-90] Yann Le Cun, John S. Denker, and Sara A. Solla. "Optimal Brain Damage." In David S. Touretzky (editor), *Advances in Neural Information Processing Systems (NIPS)*, (IEEE; Denver, Colorado; November 27–30, 1989), (ISBN: 1-55860-100-7), volume 2, pages 598–605. Morgan Kaufmann, San Mateo, California, 1990.

[Leber-92] Jean-François Leber and G. S. Moschytz. "An Acoustical Signal Recognizer Implemented on a Novel Interactive Object-Oriented Neural Network Simulator." In Igor Alexander and John Taylor (editors), *Artificial Neural Networks, 2: Proceedings of the 1992 International Conference on Artificial Neural Networks (ICANN-92)*, (Brighton, United Kingdom; 4–7 September 1992), (ISBN: 0-444-89488-8), volume 2, pages 1291–1294. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[Leber-93] Jean-François Leber. "The Recognition of Acoustical Signals Using Neural Networks and an Open Simulator." In Wolfgang Fichtner, Walter Guggenbühl, Hans Melchior, and George S. Moschytz (editors), *Series in Microelectronics*, volume 20. Hartung-Gorre Verlag, Konstanz, Germany, 1 edition, 1993.

[Lee-92] Hahn-Ming Lee and Ching-Chi Hsu. "A Neural Network Training Algorithm with the Topology Generation Ability for the Classification Problem." *International Journal of Neural Networks*, volume 3, number 1, pages 3–15, March 1992.

[Lee-93] Youngjik Lee, Sang-Hoon Oh, and Myung Won Kim. "An Analysis of Premature Saturation in Back Propagation Learning." *Neural Networks*, volume 6, pages 719–728, 1993.

[Leerink-95] Laurens R. Leerink, C. Lee Giles, Bill G. Horne, and Marwan A. Jabri. "Learning with Product Units." *Advances in Neural Information Processing Systems*, volume 7, page 537ff, 1995.

[Liang-93] Ping Liang and N. Jamali. "Artificial Neural Networks with Quasipolynomial Synapses and Product Synaptic Contacts." *Biological Cybernetics*, volume 70, number 2, pages 163–75, 1993.

[Linden-92] Alexander Linden and Christoph Tietz. "SESAME - A Software Environment for Combining Multiple Neural Network Paradigms and Applications." In Igor Alexander and John Taylor (editors), *Artificial Neural Networks, 2: Proceedings of the 1992 International Conference on Artificial Neural Networks (ICANN-92)*, (Brighton, United Kingdom; 4–7 September 1992), (ISBN: 0-444-89488-8), volume 2, pages 1265–1268. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[Linden-94] A. Linden and Ch. Tietz. "SESAME." Source and documentation obtainable via anonymous ftp from ftp.gmd.de, 1994.

[Lu-95] Bao-Liang Lu, Koji Ito, Hajime Kita, and Yoshikazu Nishikawa. "A Parallel and Modular Multi-sieving Neural Network Architecture for Constructive Learning." In *Proc. of Fourth International Conference on Artificial Neural Networks (ANN'95).* June 1995.

[Lu-96] Bao-Liang Lu and Koji Ito. "A Parallel and Modular Multi-Sieving Neural Network Architecture with Multiple Control Networks." In *Proceedings of 1996 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1303–1308. Beijing, China, October 14-17, 1996.

[Madineni-94] Kedar Babu Madineni. "Two Corner Classification Algorithms for Training the Kak Feedforward Neural Network." *Information Science*, volume 81, pages 229–234, 1994.

[Mascioli-93] F. M. Frattale Mascioli and G. Martinelli. "A Constructive Algorithm for Binary Mapping." In Stan Gielen and Bert Kappen (editors), *Proceedings of the International Conference on Artificial Neural Networks*, page 776. Springer Verlag, 1993.

[Miller-91] J. W. Miller, R. Goodman, and P. Smyth. "Objective Functions for Probability Estimation." In *International Joint Conference on Neural Networks, Seattle*, volume 1, pages 881–886. 1991.

[Minsky-69] Marvin L. Minsky and Seymour A. Papert. "*Perceptrons.*" MIT Press, Cambridge, Massachusetts, 1969.

[Mohraz-96] Karim Mohraz and Peter Protzel. "FlexNet - a Flexible Neural Network Construction Algorithm." In *European Symposium on Artificial Neural Networks (ESANN).* Bruges, Belgium, April 1996.

[Moody-96] John O. Moody and Panos J. Antsaklis. "The Dependence Identification Neural Network Construction Algorithm." *IEEE Transactions on Neural Networks*, volume 7, number 1, pages 3–15, January 1996.

[Moreira-95] M. Moreira and E. Fiesler. "Neural Networks with Adaptive Learning Rate and Momentum Terms." IDIAP-RR 4, IDIAP, C.P. 192, 1920 Martigny, Switzerland, October 1995.

[Mozer-89] Michael C. Mozer and Paul Smolensky. "Using Relevance to Reduce Network Size Automatically." *Connection Science*, volume 1, number 1, pages 3–16, 1989.

[Murphy-96] P. M. Murphy and D. W. Aha (librarians). "*UCI Repository of Machine Learning Databases.*" UCI Repository of machine learning databases, ftp access ftp.ics.uci.edu: pub/machine-learning-databases, Irvine, CA: University of California, Department of Information and Computer Science, 1996.

[Nabhan-94] Tarek M. Nabhan and Alber Y. Zomaya. "Toward Generating Neural Network Structures for Function Approximation." *Neural Networks*, volume 7, number 1, pages 89–99, 1994.

[Nguyen-90] Derrick Nguyen and Bernard Widrow. "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights." In *Proceedings of the International Joint Conference on Neural Networks (IJCNN) San Diego*, (IEEE+INNS; San Diego, California; June 17–21, 1990), volume III, pages 21–26. IEEE Neural Networks Council; Edward Brothers, Ann Arbor, Michigan, 1990.

[O'Reilly-95] Randall C. O'Reilly, Chadley K. Dawson, and James L. McClelland. "The PDP++ Software." Obtainable via anonymous ftp: hydra.psy.cmu.edu/pub/pdp++ or the URL http://www.cs.cmu.edu/Web/Groups/CNBC/PDP++/PDP++.html

[Oja-93] Erkki Oja and Juha Karhunen. "Nonlinear PCA: Algorithms and Applications." Technical Report A18, Helsinki University of Technology Laboratory of Computer and Information Sciences, Rakentajanaukio 2 C, SF-02150 Espoo, Finland, September 1993.

[Pao-89] Yoh-Han Pao. "*Adaptive Pattern Recognition and Neural Networks.*" Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[Pao-95] Y.-H. Pao and S. M. Phillips. "The Functional Link Net and Optimal Control." *Neurocomputing*, volume 9, number 2, pages 149–164, October 1995.

[Parekh-96] Rajesh Parekh, Jihoon Yang, and Vasant Honavar. "Constructive Neural Network Learning Algorithms for Multi-Category Pattern Classification." Technical Report TR95-15a, Artificial Intelligence Research Group, Department of Computer Science, 226 Atanasoff Hall, Iowa State University, Ames, IA 50011, USA, December 3, 1995.

[Parra-95] Lucas C. Parra. "Symplectic Nonlinear Component Analysis." In *Advances in Neural Information Processing Systems (NIPS)*, (Denver, Colorado; November 27 - December 2). 1995.

[Parsons-92] Ian Parsons and W. W. Armstrong. "The Use of Adaptive Logic Networks to Quantify Tar Sands Feed." Technical report, University of Alberta, September 1992. availble by anonymous ftp from ftp.cs.ualberta.ca /pub/atree/doc/alntarsands.ps.

[Platt-91] J.C. Platt. "Learning by Combining Memorization and Gradient Descent." In R. P. Lippman et al. (editor), *Advances in Neural Information Processing Systems*, volume III, pages 714–720. Morgan Kaufmann, San Mateo, 1991.

[Plaut-86] David C. Plaut, Steven J. Nowlan, and Geoffrey E. Hinton. "Experiments on Learning by Back-Propagation." Technical Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh, PA 15213, June 1986.

[Pol-95] Andrea De Pol, Georg Thimm, and Emile Fiesler. "Boolean Logic Inspired High Order Perceptron Construction." In Nicolas Droux (editor). "*SIPAR Workshop'95 Parallel and Distributed Systems.*" Rue de la Source 21, CH-2501 Biel, Switzerland, http://www.isbiel.ch/, droux@info.isbiel.ch, 1995. SIPAR SI Group for Parallel Systems, Biel School of Engineering, Computer Science Department.

[Prechelt-94] Lutz Prechelt. "PROBEN1 — A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms." Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

[Prechelt-95] Lutz Prechelt. "Adaptive Parameter Pruning in Neural Networks." Technical Report 95–009, International Computer Science Institute, Berkeley, CA, March 1995.

[Quinlan-93] R. Quinlan. "Combining Instance-Based and Model-Based Learning." In *In Proceedings on the Tenth International Conference of Machine Learning*, (University of Massachusetts), pages 236–243. Amherst. Morgan Kaufmann, 1993.

[Raina-94] Praveen Raina. "Comparison of Learning and Generalization Capabilities of the Kak and the Backpropagation Algorithm." *Information Science*, volume 81, pages 261–274, 1994.

[Redding-93] Nicholas J. Redding, Adam Kowalczyk, and Tom Downs. "Constructive Higher-Order Network Algorithm That is Polynomial Time." *Neural Networks*, volume 6, number 7, pages 997–1010, 1993.

[Reed-93] Russell Reed. "Pruning Algorithms — A Survey." *IEEE Transactions on Neural Networks*, volume 4, number 5, pages 740–747, September 1993.

[Ring-93] Mark Ring. "Learning Sequential Tasks by Incremental Higher-Order Neural Networks." Technical Report AI 93-193, Department of Computer Sciences, University of Texas at Austin, Austin, Texas, January 1993.

[Rumelhart-86] David E. Rumelhart, James L. McClelland, and the PDP Research Group. "*Parallel Distributed Processing: Explorations in the Microstructure of Cognition.*" Volume 1: Foundations. The MIT Press, Cambridge, Massachusetts, 1986.

[Sanger-91] T. D. Sanger, R. S. Sutton, and C. J. Matheus. "Iterative Construction of Sparse Polynomial Approximations." *Advances in Neural Information Processing Systems*, volume 4, pages 1064–1071, 1991.

[Sankar-91] A. Sankar and R.J. Mammone. "Improving Learning Rate of Neural Tree Networks Using Thermal Perceptrons." In *Neural Networks for Signal Processing. Proceedings of the 1991 IEEE Workshop (Cat. No.91TH0385-5)*, (IEEE; Princeton, NJ, USA; 30 Sept.-1 Oct. 1991), (ISBN: 0 7803 0118 8), pages 90–100. IEEE, New York, NY, USA, 1991.

[Saxena-94] I. Saxena and E. Fiesler. "An Adaptive Multilayer Optical Neural Network Design." Technical Report 94-04, IDIAP, C.P. 592, 1920 Martigny, Switzerland, March 1994.

[Schiffmann-92] W. H. Schiffmann, M. Joost, and R. Werner. "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons." Technical report, Institute für Physics, University of Koblenz, Koblenz, Germany, 1992.

[Schmidhuber-96] Jürgen Schmidhuber and Stefan Heil. "Sequential Neural Text Compression." *IEEE Transactions on Neural Networks*, volume 7, number 1, pages 142–146, 1996.

[Shadafan-93] M. Niranjan R.S. Shadafan. "A Dynamic Neural Network Architecture by Sequential Partitioning of the Input Space." Technical report, Cambrige University Engineering Department, Trumpington St., Cambridge, CB2 1PZ, England. Email: rss@eng.cam.ac.uk, 1993.

[Shin-95] Yoan Shin and Joydeep Ghosh. "Ridge Polynomial Networks." *IEEE Trans. Neural Networks*, volume 6, number 3, pages 610–622, May 1995.

[Shiotani-95] Shigetoshi Shiotani, Toshio Fukuda, and Takanori Shibata. "A Neural Network Architecture for Incremental Learning." *Neurocomputing*, volume 9, number 2, pages 111–130, October 1995.

[Sietsma-91] J. Sietsma and R. J. F. Dow. "Creating Artificial Neural Networks that Generalize." *Neural Networks*, volume 4, number 1, pages 67–69, 1991.

[Sigillito-89] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. "Classification of radar returns from the ionosphere using neural networks." In *Johns Hopkins APL Technical Digest*, number 10, pages 262–266. Space Physics Group, Applied Physics Laboratory, Johns Hopkins University, Johns Hopkins Road, Laurel, MD 20723, 1989.

[Śmieja-91] Frank J. Śmieja. "Hyperplane "Spin" Dynamics, Network Plasticity and Back- Propagation Learning." GMD report, GMD, St. Augustin, Germany, November 28, 1991.

[Sperduti-93] Alessandro Sperduti and Antonina Starita. "Speed Up Learning and Network Optimization With Extended Back Propagation." *Neural Networks*, volume 6, pages 365–383, 1993.

[Tenorio-90] Manoel Fernando Tenorio and Wei-Tsih Lee. "Self-Organizing Network for Optimum Supervised Learning." *IEEE Transactions on Neural Networks*, volume 1, number 1, pages 100–110, March 1990.

[Thimm-94.1] G. Thimm, R. Grau, and E. Fiesler. "Modular Object-Oriented Neural Network Simulators and Topology Generalizations." In Maria Marinaro and Pietro G. Morasso (editors), *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*, (Sorrento, Italy; 26–29 May, 1994), (ISBN: 3-540-19887-3), volume 1, pages 747–750. Springer-Verlag, London, UK, 1994.

[Thimm-94.2] Georg Thimm and Emile Fiesler. "Weight Initialization for High Order and Multilayer Perceptrons." In Marc Aguilar (editor), *Proceedings of the '94 SIPAR–Workshop on Parallel and Distributed Computing*, (SI Group for Parallel Systems), pages 87–90. Institute of Informatics University, Pérolles, Chemin du Musée 3, CH-1700 Fribourg, Switzerland, October 1994.

[Thimm-95] Georg Thimm and Emile Fiesler. "Evaluating Pruning Methods." In *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, pages A2 20–25. National Chiao-Tung University, Hsinchu, Taiwan, Republic of China, December 18-20 1995.

[Thimm-96] G. Thimm, P. Moerland, and E. Fiesler. "The Interchangeability of Learning Rate and Gain in Backpropagation Neural Networks." *Neural Computation*, volume 8, number 2, pages 451–460, February 15, 1996.

[Thrun-91] S. B. Thrun *et al.* "The Monk's Problems. A Performance Comparison of Different Learning Algorithms." Technical Report CMU-CS-91-197, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, December 1991.

[Tibshirani-96] Robert Tibshirani. "A Comparison of Some Error Estimates for Neural Network Models." *Neural Computation*, volume 8, number 1, pages 152–163, January 1996.

[Vesin-96] Jean-Marc Vesin. "*Ventricular tachyarrhythmia* data set." personal communication, 1996. Signal Processing Laboratory, Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland.

[Vinod-96] V.V. Vinod and S. Ghose. "Growing nonuniform feedforward networks for continuous mappings." *Neurocomputing*, volume 10, number 1, pages 55–69, January 1996.

[Vyšniauskas-95] Vytautas Vyšniauskas, Frans C. A. Groen, and Ben J. A. Kröse. "Orthogonal Incremental Learning of a Feedforward Network." In *Proceedings of ICANN'95*, volume 1. October 9-13, 1995.

[Watrous-93] Raymond L. Watrous and Gary M. Kuhn. "Some Considerations on the Training of Recurrent Neural Networks for Time-Varying Signals." In Marco Gori (editor), *Second Workshop on Neural Networks for Speech Processing*, (Università di Firenze; Firenze, Italy; December 10–11, 1992), pages 5–17. Edizioni LINT Trieste S.r.l., Trieste, Italy, 1993.

[Weigend-90] Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart. "Predicting the Future: a Connectionist Approach." *International Journal of Neural Systems*, volume 1, number 3, pages 193–209, 1990.

[Wen-96] Wu Wen, John Callahan, and Marcello Napolitano. "Towards developing verifiable neural network controller." In *ICTAI96 Workshop Artificial Intelligence for Aeronautics and Space*, (ONERA-CERT), pages 75–82. 2, Avenue Edouard Belin, B.P. 4025, 31055 Toulouse CEDEX, France, November 16, 1996.

[Wessels-92] Lodewyk F.A. Wessels and Etienne Barnard. "Avoiding False Local Minima by Proper Initialization of Connections." *IEEE Transactions on Neural Networks*, volume 3, number 6, pages 899–905, November 1992.

[Wolpert-96] David H. Wolpert. "The Mathematics of Search." Technical Report SFI-TR-95-02-010, The Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM, 87501, March 28, 1996.

[Wu-92] Rei-Yao Wu and Wen-Hsiang Tsai. "A Single-Layer Neural Network for Parallel Thinning." *International Journal of Neural Systems*, volume 3, number 4, pages 395–404, 1992.

[Wynne-Jones-91] Mike Wynne-Jones. "Constructive Algorithms and Pruning: Improving the Multi Layer Perceptron." In R. Vichnevetsky and J. J. H. Miller (editors), *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics*, (IMACS; Dublin, Ireland; July 22–26, 1991), volume 2, pages 747–750. IMACS International Association for Mathematics and Computers in Simulation, 1991.

[Yu-94] Xiangui Yu, Nan K. Loh, and W. C. Miller. "Training Hard-Limiting Neurons Using Back-Propagation Algorithm By Updating Steepness Factors." In *Proceedings of the IEEE International Conference on Neural Networks (ICNN94)*, (IEEE; Orlando, Florida, USA; June 27–29, 1994), (ISBN: 0-7803-1901-x), volume 1, pages 526–530. IEEE Service Center, Piscataway, NJ, USA, 1994.

[Zell-95] Andreas Zell, Günter Mamier, Michael Vogt *et al.* "SNNS 4.1." Via anonymous ftp on ftp.informatik.uni-stuttgart.de (129.69.211.2) in the directory /pub/SNNS, 1995. SNNS is (Copyright) 1990-95 University of Stuttgart.

[Zijderveld-96] P. D. Zijderveld and R. A. Vingerhoeds. “Ensuring real-time performance of expert systems.” In *ICTAI96 Workshop Artificial Intelligence for Aeronautics and Space*, (ONERA-CERT), pages 67–73. 2, Avenue Edouard Belin, B.P. 4025, 31055 Toulouse CEDEX, France, November 16, 1996.

Index