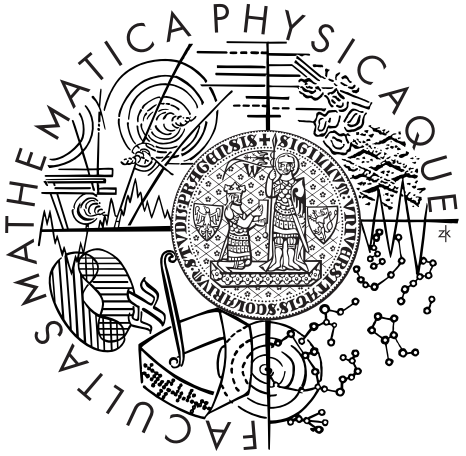
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



Zuzana Petříčková

Artificial Neural Networks and Their Usage For Knowledge Extraction

(Umělé neuronové sítě a jejich využití při extrakci znalostí)

Department of Theoretical Computer Science
and Mathematical Logic

Supervisor of the doctoral thesis: doc. RNDr. Iveta Mrázová, CSc.

Study programme: Computer Science
Specialization: Theoretical Computer Science

Prague 2015

Acknowledgments

First of all, I would like to express my sincere thanks to my supervisor doc. RNDr. Iveta Mrázová, CSc. for her extraordinarily kind, patient and tireless guidance and support throughout my whole doctoral study.

Throughout my doctoral study, my research work was partially supported by the Grant Agency of Charles University under the Grant-No. 17608, by the grant "Res Informatica" of the Grant Agency of the Czech Republic under the Grant-No. 201/09/H057, by the grant "Collegium Informaticum" of the Grant Agency of the Czech Republic under Grant-No. 201/05/H014 and by the Grant Agency of the Czech Republic under the Grants-No. 15-04960S, P202/10/133 and P103/10/0783. I thank the respective institutions for their support.

Finally, I am very grateful to my whole family for supporting me during the studies and for their help with taking care of my daughter while I was finishing this thesis. And especially, I would like to thank Markéta for her catching joyfulness and Martin for always being so patient and tolerant.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, April 9, 2015 Zuzana Petříčková

Název práce: Umělé neuronové sítě a jejich využití při extrakci znalostí

Autor: RNDr. Zuzana Petříčková

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí disertační práce: doc. RNDr. Iveta Mrázová, CSc., Katedra teoretické informatiky a matematické logiky

Abstrakt:

Vrstevnaté neuronové sítě jsou známé především díky své schopnosti dobře zobecňovat a odhalit v datech i složité nelineární závislosti. Na druhé straně má tento model tendenci vytvářet poměrně složitou vnitřní strukturu, a to především pro rozsáhlé datové sady. Při efektivním řešení náročných úloh jsou proto kladeny vysoké nároky především na rychlost procesu učení, schopnost sítě zobecňovat a na vytvoření jednoduché a transparentní struktury modelu.

V této práci jsme navrhli obecnou metodologii pro učení vrstevnatých neuronových sítí. Jejím základem je rychlá a robustní metoda škálovaných konjugovaných gradientů. Tento standardní algoritmus učení je rozšířen o analytické či aproximativní oslabování citlivosti a o vynucovaní kondenzované interní reprezentace. Redundantní vstupní a skryté neurony jsou prořezávány pomocí technik založených na citlivostní analýze a interní reprezentaci znalostí.

Vlastnosti navržené a implementované metodologie byly otestovány na řadě úloh, vesměs s pozitivním výsledkem. Vytvořený algoritmus učení je velmi rychlý a robustní k volbě parametrů. Alternativní testované metody překonává jak ve schopnosti naučených sítí zobecňovat, tak i v jejich citlivosti k šumu v datech. Metoda je schopna poměrně dobře rozpoznat irelevantní vstupní příznaky a vytvořit v průběhu učení jednoduchou a transparentní strukturu sítě. Tím usnadňuje interpretaci funkce naučené sítě.

Klíčová slova: vrstevnaté neuronové sítě, kondenzovaná interní reprezentace, citlivostní analýza, výběr příznaků, prořezávání, zobecňování

Title: Artificial Neural Networks and Their Usage For Knowledge Extraction

Author: RNDr. Zuzana Petříčková

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: doc. RNDr. Iveta Mrázová, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract:

The model of multi-layered feed-forward neural networks is well known for its ability to generalize well and to find complex non-linear dependencies in the data. On the other hand, it tends to create complex internal structures, especially for large data sets. Efficient solutions to demanding tasks currently dealt with require fast training, adequate generalization and a transparent and simple network structure.

In this thesis, we propose a general framework for training of BP-networks. It is based on the fast and robust scaled conjugate gradient technique. This classical training algorithm is enhanced with analytical or approximative sensitivity inhibition during training and enforcement of a transparent internal knowledge representation. Redundant hidden and input neurons are pruned based on internal representation and sensitivity analysis.

The performance of the developed framework has been tested on various types of data with promising results. The framework provides a fast training algorithm, robust to tunable parameters. Furthermore, it outperforms the reference techniques in the achieved generalization ability and robustness to noise in the data. It is very likely to identify redundant input features and create a simple and transparent network structure during training. In such a way it simplifies knowledge extraction from the model.

Keywords: feed-forward neural networks, condensed internal representation, sensitivity analysis, feature selection, pruning, generalization

Contents

Introduction

The subject of our study is the computational model of artificial neural networks (ANNs). Artificial neural networks represent a well-established and widely acknowledged part of Machine Learning and Artificial Intelligence.

This robust and powerful computational model was originally inspired by neurobiology — by the mechanisms and processes that are expected to operate in the human brain. Similarly to human brains, the strength of ANNs stems from the massive parallelization of simple computational elements. Furthermore, ANNs have an outstanding ability to learn from data and make reasonable predictions. This makes them suitable to solve even complex tasks from many emerging areas. Until now, the model has been successfully applied to a wide range of real-world tasks like signal processing [92], time series analysis, bioinformatics, data and web mining or multimedia information processing.

In this thesis, we concentrate on the computational model of fully-connected multilayer feed-forward neural networks (BP-networks). When compared to other neural network and machine learning models (e.g., decision trees, linear regression), the model of BP-networks is appreciated for a relatively simple topology and robustness against noise and other problems of the data (e.g., outliers, high dimensionality). Further advantages of BP-networks are their superior approximation and generalization abilities and fast response once the model is trained. And above all, the model is very likely to find complex non-linear dependencies in the data.

In our research, we have worked a lot with real-world economical data obtained from the World Bank [109]. This data set comprises various economical and demographical characteristics of the particular countries – the so called World development indicators (WDI-indicators).

An interesting property of the *World Bank* data is, that it contains various mutual relationships among the input features. Some of the relationships are linear or logarithmic and therefore discoverable using the standard, mostly linear, methods (e.g., linear regression, PCA [53] or correlation coefficients [41]). Such simple relationships have been studied a lot by economic researchers in the past years. For example, Baird et al. in [9] found a negative linear relationship between Gross domestic product per capita and Infant mortality.

In addition to relatively simple mutual relationships, the *World Bank* data contains also complex non-linear dependencies, that are very difficult to discover by means of standard methods. The process of their extraction is thus very time-consuming and with no guarantee of success.

An example of a more complex mutual relationship between the WDI-indicators is shown in Figure 1. The figure depicts the values of the following input features: Fertility rate, Number of fixed line and mobile phone subscribers, Number of Internet users and Life expectancy at birth. We can clearly see, that these WDI-indicators give a very similar information. If we knew for a single country the value of just one of them, we could manually predict the others based on the graph. However, the mutual relationship among these four features is not simple enough to be detected using the above-listed standard methods. On the other hand, the computational model of BP-networks represents a valuable option, as it is very powerful in modeling even more complex relations.

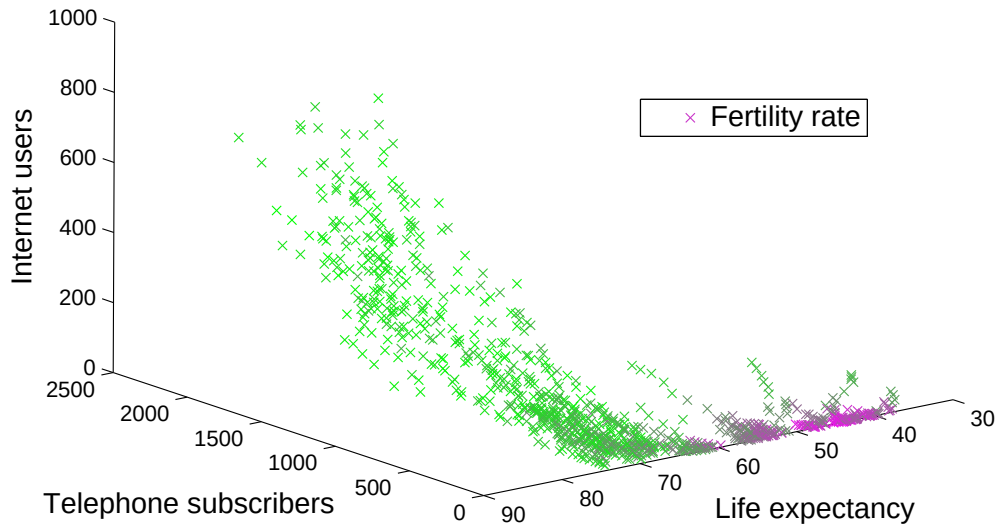


Figure 1: Mutual relationship among the following WDI-indicators: Fertility rate, Number of fixed line and mobile phone subscribers, Number of Internet users and Life expectancy at birth.

To summarize our motivation example, the main objective of this thesis is to quickly extract complex non-linear dependencies present in the data and to describe them in a comprehensible way. To solve this task, we use the powerful and robust computational model of BP-networks.

On one hand, BP-networks are remarkably better in finding complex dependencies in the data than classical statistical models (e.g., decision trees or linear regression). On the other hand, they have a tendency to create relatively complex internal structures, especially for large and demanding tasks. In such a case, it may be difficult to describe the extracted knowledge in a simple way (e.g., in the form of rules). Complex tasks also have greater demands on the generalization ability of the BP-network model that the standard training algorithm (back-propagation) can only hardly achieve or it becomes extremely slow.

Fortunately, there are several enhancements of the standard BP-model and its training process, that overcome the above-sketched difficulties. Based on their main purpose, the enhancing techniques can be divided into the following categories: fast training, topology simplification, creation of a transparent network structure with greater expressive power and improvement of generalization ability. In the following paragraphs, we will provide a brief summary of these techniques.

When considering the techniques for faster training of BP-networks, we can choose among relatively simple enhancements of the standard back-propagation algorithm (e.g., learning with momentum, early stopping, adaptive learning rate methods [52, 99]) and sophisticated training techniques (e.g., second order or conjugate gradient optimization methods [14, 35, 71, 75, 91]). Especially the conjugate gradients methods (e.g., the Scaled conjugate gradients training algorithm (SCG) [75]) offer fast training and robustness to noise and outliers in

the data. They are appreciated for both adequate generalization and low space complexity.

While most of the standard training algorithms work with a fixed topology, structure optimization methods try to find the optimal network topology automatically during training. In such a way they simplify further knowledge extraction from the model. The key representatives of structure optimization methods are pruning algorithms [28, 34, 43, 62, 77, 126], regularization techniques [36, 111, 119] and network construction methods [33, 66]. Some of the pruning techniques also try to automatically identify relevant input features [29].

BP-networks represent knowledge in a distributed way. Although they tend to form relatively unclear internal structures, there are regularization techniques that help to make the internal structure more transparent (e.g., the methods of learning condensed or unambiguous internal representation [86]). The main purpose of these techniques is to improve the expressive power of the BP-network model.

Among the techniques that improve the generalization ability of the trained BP-networks, we may highlight especially learning from hints [2], training with jitter [73, 93, 104], cross-validation and early stopping [95], and the methods for structure optimization. Most of the above-listed methods also try to reduce the VC-dimension of the final network structures and make the network function smoother [114].

In Chapter 1, we state the basic concepts used in the thesis. We define formally the model of artificial neural networks and describe the standard back-propagation training algorithm. After that, we provide a brief analysis of this model and its training algorithm.

In Chapter 2, we state and discuss the main goals of this thesis. In Chapter 3, we continue with a thorough description of existing approaches to our goals. We structure Chapter 3 based on the following issues: fast training, feature selection, structure optimization, generalization improvement and transparency.

In Chapter 4, we propose a general framework for training of BP-networks. We introduce three successive versions of our framework (SCGIR, SCGS and SCGSA) that differ primarily in the regularization techniques included. Chapter 5 is devoted to experimental evaluation of the designed framework. The chapter is structured based on the following fields of interest – generalization, speed, transparency and structure optimization.

1. Basic concepts

In this chapter we will describe basic concepts used in the thesis. We will define formally the model of artificial neural networks and state the corresponding terminology used in the area of neural networks (Reed [95], Bishop [16], [15], Rojas [100] and Haykin [48]).

1.1 Formal neuron

An artificial neural network (ANN) is a computational model that consists of simple computational units called formal neurons (or just neurons) [101]. It was originally inspired by neurobiology – by the mechanisms and processes that are expected to operate in the human brain. Similarly to human brains, the strength of ANNs stems from the massive parallelization of simple computational elements. Furthermore, both types of neural networks (the biological as well as the artificial ones) have an outstanding ability to learn from the data and make reasonable predictions. This makes them suitable to solve even complex tasks such as signal processing [92].

The biological neuron is the elementary part of the nervous system. Its main purpose is to transfer and process signals – electric and electrochemical impulses. It consists of three main parts:

- dendrites that transfer the input signals into the neuron,
- cell body, which accumulates and processes the signal,
- and axon that transmits the output signal to other neurons.

The axon is connected to dendrites of other neurons. The neuron accumulates the input signals weighted by the strength of the corresponding dendrites. If the summed electric impulse exceeds a given threshold, the neuron becomes active – it generates the electric impulse and transfers it to other neurons via the axon. The repeated excitation of a neuron magnifies the strength of the respective connections and so the neural network learns.

The model of a formal neuron was inspired by the biological neuron.

Definition 1. A formal neuron is a computational unit with the weight vector $(w_1, ..., w_n) \in \Re^n$, the threshold $h \in \Re$ and the transfer function $f : \Re \to \Re$. For an arbitrary input vector $\vec{x} \in \Re^n$ the neuron computes its output $y \in \Re$ as the value of the function f at ξ. ξ denotes the value of the inner potential of the neuron:

$$\xi = \sum_{i=1}^{n} w_i x_i - h. \tag{1.1}$$

The output value $y = f(\xi)$ is called the activity of the neuron.

We will use the following formal modification of the definition of ξ in order to simplify the notation in the further formulas:

$$\xi = \sum_{i=0}^{n} w_i \, x_i, \tag{1.2}$$

where $x_0 = 1$ and $w_0 = -h$ (the thresholds are represented by weights coming from fictive neurons with a constant output value 1). The structure of the formal neuron is shown in Figure 1.1.

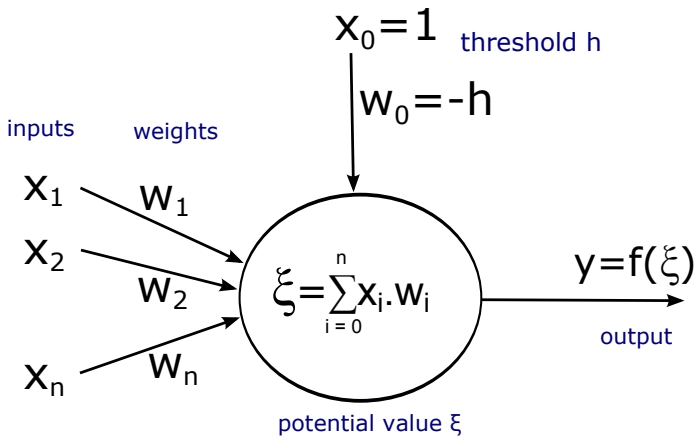


Figure 1.1: A formal neuron.

Transfer functions The functionality of a neuron depends on the choice of the transfer function. Table 1.1 briefly summarizes some of the basic and most common variants of the transfer functions: step, signum, saturated linear, linear, sigmoidal and hyperbolic tangent functions. Some of the training algorithms described later in this work, such as back-propagation (sec. 1.4), put some further restrictions on the transfer functions, e.g., differentiability. For this reason, Table 1.1 shows also the derivatives of the functions that are differentiable on $\Re$. The graphs of the sigmoidal and hyperbolic tangent functions and their derivatives are depicted in Figure 1.2.

Table 1.1: Transfer functions and their derivatives.

name		function value	derivative
step	$f : \Re \to \{0, 1\}$	$f(\xi) = \begin{cases} 1, & \xi > 0 \\ 0, & \xi \leq 0 \end{cases}$	–
signum	$f : \Re \to \{-1, 0, 1\}$	$f(\xi) = \begin{cases} 1, & \xi > 0 \\ 0, & \xi = 0 \\ -1, & \xi < 0 \end{cases}$	–
saturated linear	$f : \Re \to < 0, 1 >$	$f(\xi) = \begin{cases} 1, & \xi > 1 \\ \xi, & 0 \leq \xi \leq 1 \\ 0, & \xi < 0 \end{cases}$	–
linear	$f : \Re \to \Re$	$f(\xi) = \xi.$	$\frac{\partial y}{\partial \xi} = f'(\xi) = 1$
sigmoidal	$f : \Re \to (0, 1)$	$f(\xi) = \frac{1}{1+e^{-\lambda\xi}},\ \lambda > 0$	$\frac{\partial y}{\partial \xi} = f'(\xi) = \frac{\lambda e^{-\lambda\xi}}{(1+e^{-\lambda\xi})} = \lambda y(1-y)$
hyperbolic tangent	$f : \Re \to (-1, 1)$	$f(\xi) = \frac{1-e^{-a\,\xi}}{1+e^{-a\,\xi}}, a > 0$	$\frac{\partial y}{\partial \xi} = f'(\xi) = \frac{2a\,e^{-a\,\xi}}{(1+e^{-a\,\xi})^2} = \frac{a}{2}(1+y)\,(1-y)$

1.2 Artificial neural network

The artificial neural network is a computational model that consists of neurons, which are mutually interconnected so that the output of each neuron can serve as the input of one or more other neurons. More formally:

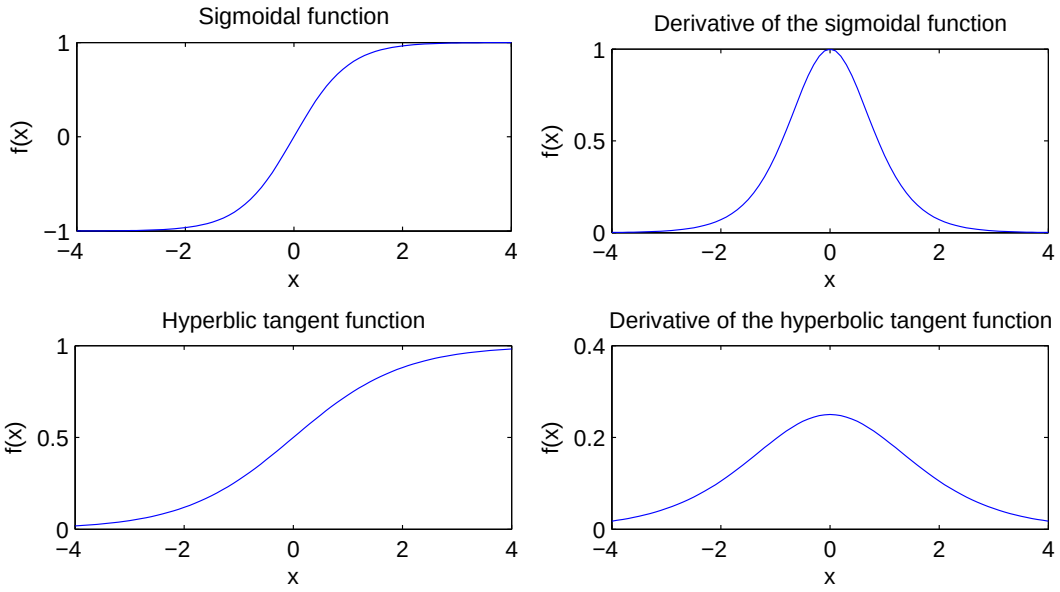


Figure 1.2: Graphs of the sigmoidal and hyperbolic transfer functions and their derivatives on the interval $[-4, 4]$.

Definition 2. The artificial neural network (ANN) is a tuple $M = (N, C, I, O, w, t)$, where

- N is the non-empty finite set of neurons,
- $C \subseteq N \times N$ is the non-empty set of oriented edges that connect the pairs of neurons,
- $I \subseteq N$ is the non-empty set of input neurons,
- $O \subseteq N$ is the non-empty set of output neurons,
- $w : C \to \Re$ is the weight function,
- $t : N \to \Re$ is the threshold function.

The topology of a neural network is given by the number of neurons and their mutual interconnections.

Definition 3. The topology of the ANN $M = (N, C, I, O, w, t)$ is an oriented graph with the set of nodes N and the set of edges C. The edges are weighted by w.

There exist various topologies of ANNs. The so-called recurrent neural networks contain cycles. Acyclic ANNs are called feed-forward. For a feed-forward neural network, the input neurons are the neurons with only outgoing edges, while the output neurons have only incoming edges. If the neurons are arranged into multiple layers, we name the model multilayer. In the past years, a strong effort has been made in the field of so-called deep networks (introduced by LeCun et al. in [65]).

In this work, we will concentrate on the ANNS with the layered, fully-connected and feed-forward topology.

Definition 4. The fully-connected multilayer feed-forward neural network (BP-network) is an ANN that fulfills the following requirements:

- The set of edges C together with the set of nodes N form an acyclic oriented graph.
- The set N consists of the sequence $L_1, ..., L_{l+2}$ of $(l+2)$ disjoint subsets called layers.
- Each neuron in the layer L_k is connected to all neurons of the subsequent layer L_{k+1}, $k = 1, ..., l+1$. C contains only edges from the k-th to the $(k+1)$-th layer, $k = 1, ..., l+1$.
- The first layer, called the input layer, is the set of n input neurons. The input neurons have just one input and their transfer function is linear – identity.
- The last layer, denoted as the output layer, consists of m output neurons.
- All other neurons are called hidden neurons. They are contained in the remaining l, hidden, layers $L_2, ..., L_{l+1}$.

Generally, each neuron in the BP-network can has its own transfer function. In this work, we will use the model, where all the hidden neurons have the hyperbolic tangent transfer functions, while all the output neurons implement the linear transfer functions.

The topology of a BP-network is usually a priori given and fixed. It can be described by the term $(l_1 = n) - l_2 - ... - l_{l+1} - (l_{l+2} = m)$, where l_k is the number of neurons in the layer L_k, $k = 1, ..., l+2$ and the sign "–" is the delimiter. Fig. 1.3 shows an example BP-network with the topology 4-3-4-2.

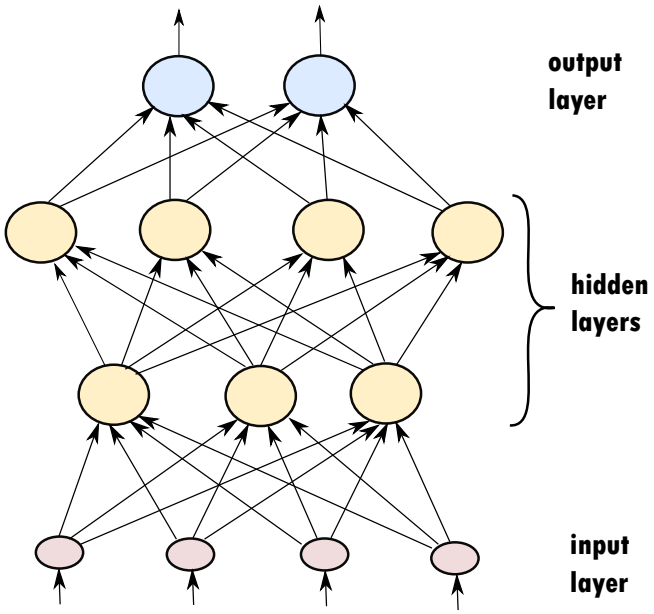


Figure 1.3: Example of the 4-3-4-2 BP-network topology.

Another important concept of the BP-networks is the configuration.

Definition 5. The configuration of the ANN is given by all the weights (and thresholds) in the ANN. It can be expressed by the weight vector $\vec{w}$ of size W:

$$\vec{w} = (w_{ij})_{[i,j]\in C}\ (t_k)_{k\in N}, \tag{1.3}$$

where W is the total number of weights and thresholds.

The function of the BP-network can be described by two different processes – the recall process and the training process. We will describe them in more detail in the following subsections.

1.2.1 Recall process

Each BP-network implements a network function $\varphi : \Re^n \to \Re^m$. If we present an input vector $\vec{x} \in \Re^n$ to the model, it will compute the output vector $\vec{y} = \varphi(\vec{x}) \in \Re^m$ in the following way: If all the hidden and output neurons have the same transfer function f, the j-th element of the output vector corresponding to the j-th output neuron is given by:

$$\varphi(\vec{x})_j = f(\sum_{i\in L_{l+1}} w_{ij}\ f(...f(\sum_{i_1\in L_1} w_{i_1 i_2}\ x_{i_1})...)), \tag{1.4}$$

where i, i_1 and i_2 index the neurons in the layers L_{l+1}, L_1 and L_2, respectively. The recall process has the following schema:

1. Present an input vector $\vec{x} \in \Re^n$.

2. For $i = 1, ..., n$, the activity of the i-th input neuron is set to the i-th element of the input vector, $y_i = x_i$.

3. For each network layer $L_k,\ k = 2, ..., l+2$ (successively from the first hidden layer to the output layer) and for each neuron j in the layer L_k compute its potential with the use of the activities of the neurons in the preceding layer (indexed by i):

   $$\xi_j = \sum_{i\in L_{k-1}} w_{ij}\ y_i. \tag{1.5}$$

   Then compute the activity of the neuron j:

   $$y_j = f(\xi_j) = f(\sum_{i\in L_{k-1}} w_{ij}\ y_i), \tag{1.6}$$

   where f is the transfer function.

4. At the end of the recall process, the activities of all neurons are computed. The actual network output $\vec{y} \in \Re^m$ is given by the activities of the output neurons.

1.2.2 Training process

The objective of the training process of a BP-network is to approximate an sought after function as well as possible. The knowledge is stored in the network weights and thresholds, i.e., in the configuration. A natural way to train BP-networks is so-called supervised training, i.e., learning from examples. That means, that the wanted network function is unknown, its value is given just for a training set T of P input-output patterns:

$$T \;=\; \{\,(\,\vec{x}_p, \vec{d}_p\,)\,|\quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1,\ldots,P\}\}. \tag{1.7}$$

for a network with n input neurons and m output neurons. The input patterns $\vec{x}_p$ are the input vectors for the BP-network, while the output patterns $\vec{d}_p$ are the desired network outputs. The actual outputs produced by the network will be denoted by $\vec{y}_p \in \Re^m$. The elements $x_{p1}, x_{p2}, ..., x_{pn}$ of the input pattern $\vec{x}_p$ are called the input features. The dimensionality of the input data is defined as the number n of input features.

At the end of the training process, the actual output for each input pattern should be as close as possible to the corresponding output pattern. Any difference is treated as an error to be minimized. The actual network behavior can be evaluated by the error function $E = E(\vec{w})$, where $\vec{w}$ is the current configuration. The standard performance measure is the sum of squared errors E_{SSE}:

$$E(\vec{w}) \;=\; E \;=\; E_{SSE} \;=\; \sum_{p=1}^{P} E_p \;=\; \frac{1}{2}\sum_{p=1}^{P} \|\,\vec{y}_p - \vec{d}_p\,\|^2 \;=\; \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{m} (\,y_{pj} - d_{pj}\,)^2 \tag{1.8}$$

where p is an index over all training patterns, j indexes all output neurons, and E_p is the error corresponding to the p-th training pattern. The standard error function E_{SSE} can be replaced by alternative performance measures. A common option is E_{MSE} (mean square error), defined as:

$$E_{MSE} \;=\; \frac{1}{mP} E_{SSE} \;=\; \frac{1}{2\,m\,P}\sum_{p=1}^{P}\sum_{j=1}^{m} (\,y_{pj} \;-\; d_{pj}\,)^2, \tag{1.9}$$

where m and P are the numbers of output neurons and patterns, respectively. E_{MSE} is advisable for large training sets and networks with a large number of outputs. The error function can also be altered by the so-called penalty terms, reflecting various further training goals added to the performance (e.g., restrictions on the network function).

The error function $E(\vec{w})$ can be at the point $(\vec{w} + \vec{z})$ expressed by the Taylor series:

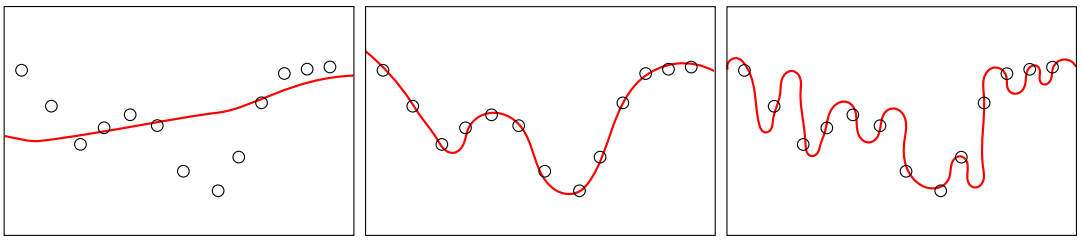
$$E(\vec{w} + \vec{z}) = E(\vec{w}) + E'(\vec{w})^T \vec{z} + \frac{1}{2}\vec{z}^{\,T} E''(\vec{w})\vec{z} + \cdots, \tag{1.10}$$

where $E'(\vec{w})$ is the gradient vector of the length W with the elements $E'(\vec{w})_i = \frac{\partial E}{\partial w_i}$, and $E''(\vec{w})$ is the $W \times W$ Hessian matrix of second derivatives with the elements $E''(\vec{w})_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$, W is the lenght of $\vec{w}$.

1.3 Generalization and VC-dimension

Generalization is the ability of a computational model to give correct outputs not only to the patterns in the training set, but also to previously unseen or noisy input patterns.

Poor generalization is often related to the problem of so called overtraining. If a computational model is overtrained, it tends to memorize the training patterns and is not able to recognize well patterns outside of the training set. The opposite problem to overtraining is called undertraining. A model is undertrained, if it is too simple and it doesn't fit the training data. Figure 1.4 illustrates both problems of undertraining and overtraining.



(a) (b) (c)

Figure 1.4: Generalization ability of a computational model. Training patterns are indicated by black circles, model function is indicated by a red line. (a) undertrained model (too simple approximation), (b) fitting model (good generalization), (c) overtrained model (poor generalization).

Generalization error In practice, the generalization ability can be roughly assessed as the performance error on the test set of patterns (patterns not present in the training set or noisy patterns). To achieve reliable results, the test set should be sufficiently large and mimic the actual data distribution.

Vapnik–Chervonenkis dimension The Vapnik-Chervonenkis dimension (or VC-dimension) was originally defined by Vapnik and Chervonenkis [113]. It measures the complexity of a learning system as the maximum number of points that the learning system (i.e., the class of concepts) can shatter. The higher is the VC-dimension of a learning system, the more training patterns are needed to achieve an acceptable generalization. A learning system is called *learnable*, if its VC-dimension is finite.

1.4 Back-propagation algorithm

The back-propagation training algorithm (BP-algorithm) is one of the most commonly used concept for training BP-networks. The algorithm was independently discovered by several authors (Werbos [118], Parker [89], LeCun [63] and Rummelhart, Hinton and Williams [102]), however it became generally known mostly thanks to the last contribution [102].

BP-algorithm is an optimization method over the network weights and thresholds – the aim of training is to find a set of weights and thresholds that approximate the wanted network function as well as possible. It is designed for supervised training, i.e. it learns from examples. It uses the training set $T = \{ (\vec{x}_p, \vec{d}_p) \mid \quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$ (1.7), for a network with n input neurons and m output neurons. As the performance measure, the standard BP-algorithm uses the sum of squared errors function $E = E_{SSE} = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{m} (y_{pj} - d_{pj})^2$ (1.8), where p and j are the indexes over all the training patterns and all the output neurons, respectively. Vectors $\vec{x}_p$, $\vec{y}_p$ and $\vec{d}_p$ are the input patterns, the actual outputs and the desired output patterns, respectively.

BP-algorithm is a gradient method. In each step we choose a direction, in which the error function decreases the most and adjust the weights in this direction (against the gradient of the error function). The optimization process is iterative and proceeds in discrete time steps. The basic schema of an iteration follows:

1. Select a training pattern from the training set.

2. *Forward propagation*: Present the selected training pattern to the network and compute the actual network output (in the direction from the input to the output layer).

3. Compare the actual and desired output and compute the error.

4. *Back-propagation*: Adjust the weights to minimize the error (backwards from the output to the input layer).

The process is repeated until the stop condition is satisfied – for example until the error is reasonably small or until a pre-set (maximal) number of iterations is reached.

In the following paragraphs, we will briefly describe the derivation of the adaptation rules for the described algorithm. We assume that all the network transfer functions are continuous and differentiable. The weights and thresholds of the network w are adjusted iteratively after presenting each respective training pattern $(\vec{x}_p, \vec{d}_p)$ by:

$$w_{ij}(t+1) \; = \; w_{ij}(t) + \Delta w_{ij}(t), \tag{1.11}$$

where $\Delta w_{ij}(t)$ is the change of w_{ij} in time t. $t+1$ and t index next and present weights, respectively.

$\Delta w_{ij}(t)$ is proportional to the negative partial derivative of the pattern error function:

$$\Delta w_{ij}(t) \; \simeq \; -\alpha \, \frac{\partial E_p}{\partial w_{ij}}, \tag{1.12}$$

where α is a positive constant representing the learning rate – it defines the step length in the negative gradient direction and controls the speed of convergence. In the following terms, the actual output value and the potential of a neuron j will be denoted as y_{pj} and ξ_{pj}, respectively. We apply the chain rule for partial derivatives and obtain:

$$\frac{\partial E_p}{\partial w_{ij}} \;=\; \frac{\partial E_p}{\partial y_{pj}}\,\frac{\partial y_{pj}}{\partial \xi_{pj}}\,\frac{\partial \xi_{pj}}{\partial w_{ij}} \;=\; \frac{\partial E_p}{\partial y_{pj}}\,f'(\xi_{pj})\,y_{pi} \;=\; \delta_{pj}\,y_{pi}, \tag{1.13}$$

where $\delta_{pj} = \frac{\partial E_p}{\partial y_{pj}}\,\frac{\partial y_{pj}}{\partial \xi_{pj}}$ is a useful notation often referred to as back-propagated error. For an output neuron j, the derivative can be computed directly:

$$\delta_{pj} \;=\; (y_{pj} - d_{pj})\,f'(\xi_{pj}). \tag{1.14}$$

For a hidden neuron j, $\frac{\partial E_p}{\partial y_{pj}}$ will be computed indirectly using the chain rule:

$$\frac{\partial E_p}{\partial y_{pj}} \;=\; \sum_q \frac{\partial E_p}{\partial \xi_{pq}}\,\frac{\partial \xi_{pq}}{\partial y_{pj}} \;=\; \sum_q \delta_{pq}\,w_{jq}, \tag{1.15}$$

$$\delta_{pj} \;=\; f'(\xi_{pj}) \sum_q \delta_{pq}\,w_{jq}, \tag{1.16}$$

where q indexes neurons in the layer above the neuron j.

Altogether:

$$w_{ij}(t+1) \;=\; w_{ij}(t) - \alpha\,\delta_{pj}\,y_{pi}, \tag{1.17}$$

$$\delta_{pj} \;=\; \begin{cases} (\,y_{pj} - d_{pj}\,)\,f'(\xi_{pj}) & \text{for an output neuron,} \\ f'(\xi_{pj}) \sum_q \delta_{pq}\,w_{jq} & \text{for a hidden neuron.} \end{cases} \tag{1.18}$$

Algorithm 1.1 summarizes the steps of the BP-algorithm for the BP-network model with the hyperbolic tangent transfer function in the hidden layers and the linear transfer function in the output layer.

The behavior of the BP-algorithm varies depending on how the details are dealt with. For example, we can choose among various simple or sophisticated weight-initialization strategies [110]. An example of a simple heuristic is to select the weights randomly from an interval such as $(-\frac{2}{\sqrt{n}}, \frac{2}{\sqrt{n}})$, where n is the number of inputs. A more sophisticated possibility is e.g. the Nguyen-Widrow initialization technique [87] that forces the initial weight vectors to cover well the input space of each layer.

Because the BP-algorithm is not guaranteed to converge, there are various stop conditions, which are in applications usually combined. The basic stop criterion is a given maximal number of epochs / iterations or a time limit. Another possibility is to stop training if the average weight change $\Delta w_{ij}(t)$ falls under a pre-determined lower bound, i.e., if the adaptation process is too slow.

The stop criterion can also be based on the error function – we can define the requested value of network error (averaged over all training patterns and network outputs) on the training set or on previously unseen (validation) data. If the desired error value is reached, the training is stopped. We can similarly use also the pre-determined maximal number of consecutive epochs/iterations in which the training/validation error grows.

Algorithm 1.1 Back-propagation algorithm

0. *Input:*
   BP-network $M = (N, C, I, O, w, t)$,
   Training set $T = \{ (\vec{x}_p, \vec{d}_p) \mid \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$,
   Stop criteria, e.g.:
   a. The training or validation error is lower than a pre-set value.
   b. The time index t has reached the pre-determined value.
   c. The average validation error grows in a more consecutive iterations than allowed.
1. *Initialization:*
   Initialize all the weights and thresholds with small random values.
   Set discrete time index t to 0.
2. Present a newly chosen training pattern in the form $(\vec{x}_p, \vec{d}_p)$.
3. *Forward propagation:*
   Compute the actual network output $\vec{y}_p$:
   For each network layer L (in the direction from the input to the output layer) and for each neuron j in layer L compute its potential and activity with the use of activities of neurons in the preceding layer (indexed by i).
   $\xi_{pj} = \sum_i w_{ij}\, x_{pi}$,
   $y_{pj} = f(\xi_{pj})$
4. *Back-propagation:*
   4a. Compute the value of back-propagated error:
   For each network layer L (in the direction from the output to the input layer) and for each neuron j in layer L compute the value of δ_{pj} with the use of actual values of δ_{pq} of neurons from the following layer (indexed by q):

$$\delta_{pj} = \begin{cases} y_{pj} - d_{pj} & \text{for an output neuron j,} \\ (1 + y_{pj})(1 - y_{pj}) \sum_q \delta_{pq}\, w_{jq} & \text{for a hidden neuron j.} \end{cases}$$

   4b. Adjust each network weight w_{ij} (from the neuron i to neuron j):
   $w_{ij}(t+1) = w_{ij}(t) - \alpha\, \delta_{pj}\, y_{pi}$.
5. If the *stop criteria* are not satisfied, set $t = t + 1$ and continue with step 2.

Another issue is the ordering of training patterns. A common strategy, sometimes called *online learning*, is to select the patterns randomly, while each particular pattern can be presented once or in more consecutive iterations. Other variant is to present the whole training set systematically in cycles called epochs. This is typical for *batch learning*, where the weights and thresholds are adjusted only once per epoch after presenting all the training patterns. The formula 1.17 is in such a case modified in the following way:

$$w_{ij}(c+1) \;=\; w_{ij}(c) - \alpha \sum_{p=1}^{P} \frac{\partial E_p}{\partial w_{ij}} \;=\; w_{ij}(c) - \alpha \sum_{p=1}^{P} \delta_{pj}\, y_{pi}, \tag{1.19}$$

where p and c index the training patterns and epochs, respectively. For batch learning, the adaptation process is usually smoother but slower than for the standard algorithm (it follows the gradient more precisely, however each training pattern weights less). The method is supposed to converge better and be more stable, however it may have a bigger chance of getting stuck in a poor local minimum of the error function [95].

Although we refer to the multi-layered fully connected feed-forward neural networks as to BP-networks, other optimization methods can be used to update the weights and thresholds of this model as well (e.g., second-order gradient techniques [12], genetic algorithms or simulated annealing [37]). On the other hand, the BP-algorithm is not restricted to the BP-networks, the Algorithm 1.1 can be easily extended to handle with different (i.e., not layered) network topologies.

1.5 Analysis of the BP-algorithm and the standard BP-model

In this section, we will discuss the main advantages and drawbacks of the standard BP-model and the BP-algorithm.

1.5.1 Analysis of the standard BP-model

The computational model of BP-networks is very efficient and robust, with remarkable approximation and generalization abilities. The tasks suitable to BP-networks usually involve function approximation or pattern recognition. In the past years, the model has been successfully applied to a wide range of real-world problems (including signal processing, prediction and control systems, bioinformatics, data compression or web mining).

When compared to other neural network and 'general' machine learning models (e.g., decision trees, linear regression), the model of BP-networks is appreciated for its simple topology, robustness against the noise and other problems of the data (outliers, course of dimensionality) and fast response once the model is trained.

In the following paragraphs, we will discuss the following basic questions: a) what kind of functions can be approximated by the BP-networks, b) what is the expected computational complexity of the learning problem, and c) the question of learnability of this learning system.

Approximation Cybenko [25], Hornik [50] and others formulated and proved the ability of universal approximation for BP-networks. The universal approximation theorem states that BP-networks with one hidden layer (and meeting some additional conditions) are universal approximators for the class of real continuous functions on compact subsets of $\Re^n$. In detail, Cybenko [25] proved, that any real continuous function $f : [0,1]^n \rightarrow (0,1)$ can be approximated arbitrarily close by a BP-network with one hidden layer, sigmoidal transfer function in the hidden layer and linear transfer function in the output layer, while the BP-network might require any number of hidden neurons [25]:

Theorem 1. Let σ be the sigmoidal function and $C(I_n)$ the space of continuous functions on $[0,1]^n$. Given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum $G(x)$ of the form

$$G(\vec{x}) = \sum_{j=1}^{N} a_j \sigma(\vec{w}_j{}^T \vec{x} + h_j) \; ; \; \vec{w}_j \in \Re^n, a_j, h_j \in \Re, j = 1, ..., N \tag{1.20}$$

for which

$$|G(\vec{x}) - f(\vec{x})| < \epsilon \text{ for all } \vec{x} \in [0,1]^n. \tag{1.21}$$

Hornik [50], Kůrková [59] and others generalized this result to networks with other types of transfer functions in the hidden layer and for other classes of ANNs.

Based on the universal approximation theorem, it may seem like the BP-network topology with one hidden layer is always a sufficient choice. However, architectures with more hidden layers are more appropriate for many tasks – we may need less computational units in sum, the training could be easier and faster and the final network structure may be more transparent.

Computational complexity of the learning problem Judd [54] showed that the general learning problem for BP-networks with the step transfer function is NP-complete. That means, that no algorithm is known, that will solve the general learning problem for BP-networks with the step transfer function in polynomial time. In the worst case, the time grows exponentially with the problem size (i.e., with the number of weights and thresholds) [100]. Šíma [107] proved the NP-hardness of the learning problem even for the minimal BP-network topology with a single neuron with the sigmoidal transfer function. However, some restricted architectures of BP-networks can be trained in a polynomial time [54].

These theoretical results show, that it makes sense to use approximative methods for BP-training (e.g., back-propagation) and to search for heuristics that will speed up the training process.

Learnability and VC-dimension The higher is the complexity of the learning problem, the more training patterns are needed to achieve sufficient model performance and generalization. This issue is related to the VC-dimension and learnability of the BP-networks (see Section 1.3). Baum and others [13] showed, that the VC-dimension d^{VC} of a BP-network with N neurons and W weights can be estimated as:

$$d^{VC} \leq 2\, W\, log_2(e\, N), \tag{1.22}$$

where e is the base of the natural logarithm. To achieve a given generalization error rate $0 < \epsilon \leq 1/8$, the needed number P of randomly selected training patterns can be bounded as [13]:

$$P \geq O(\frac{W}{\epsilon} log_2 \frac{N}{\epsilon}). \tag{1.23}$$

High VC-dimension is often connected with worse generalization. The VC-dimension of BP-networks can be reduced, e.g., by learning from hints [2], regularization or pruning. If we reduce the number of degrees of freedom of the BP-network, we also reduce the inherent complexity of the learning problem [100].

1.5.2 Analysis of the BP-algorithm

Although there exist many more sophisticated training algorithms for BP-networks (and even more sophisticated neural network models), the standard back propagation method is still quite popular in real-world applications. The main reasons may be, that it is simple to implement and easy to understand, well-documented and tested. Once the BP-network is trained, the response is also very fast.

On the other side, the main disadvantages of the standard BP-algorithm are the low speed of the training process (with a tendency to get stuck in local minima of the error function), its high sensitivity to the choice of initial parameters (especially topology) and low transparency of the final model. In the following paragraphs, we will discuss these topics in more detail.

Time costs Due to recurrent computations of the back-propagated error δ_{pj} , the time complexity of one iteration of the BP-algorithm is $O(W^2)$, where W is the number of weights (and thresholds) in the network. Nevertheless, for many even simple real applications the training process may be slow – especially in terms of the number of iterations and the total training time. The low speed is partly a coincidence of the simplicity of the algorithm, but it may be caused also by other factors, such as bad choice of the topology and other model parameters.

Furthermore, the training set should be prepared carefully. It should be well-balanced, represent adequately the desired network function and cover the input space as well as possible. If the dimensionality of the input data is high, more training patterns are needed to achieve good model performance and generalization. Poor data representation with a larger number of irrelevant inputs may lead to slow convergence and may even prevent the BP-network from learning the task. To accelerate the training process, it is also important to normalize both the input and output patterns. According to LeCun [64], the input variables should be uncorrelated.

Long training times can be also caused by paralysis caused by sigmoid saturation. The back-propagated error δ_{pj} is proportional to the derivative of the transfer function $f'(\xi_j)$. If the potentials ξ_j are (absolutely) large, the values of $f'(\xi_j)$ (and δ_{pj}) are almost zero (it is caused by flat tails of sigmoidal and similar transfer functions). A neuron is saturated, if this happens for most of the training patterns. Input weights of a saturated neuron stop adapting. If there is a larger

number of saturated neurons in a BP-network, the training becomes extremely slow or it can even stop in a poor state.

Large potentials can be caused by great input weights or inputs. Therefore, the problem of paralysis can be significantly reduced by normalization of the input and output patterns – the desired outputs should be within the range of the corresponding transfer function. It is also advisable to initialize the weights adequately with small random values and to use an antisymmetric transfer function (such as hyperbolic tangent) [48], [95]. Further possibility is to detect and resolve the paralysis during training [116] (e.g., by reducing the training rate or by scaling the weights).

The speed of training and even the convergence depends on a careful choice of the learning rate $0 < \alpha \leq 1$. If the value is too high, sufficient minima of the error function can be skipped or the algorithm may tend to oscillate around the minimum. If α is too small, the training will become extremely slow (and it can more easily get stuck in a poor local minimum of the error function).

The optimal choice of α depends not only on the given task, but also on other aspects, such as the current position in the weight space and the error surface. In flat regions of the error surface, the value of α could be higher, while in steep regions it should be lower. Faster training can be achieved (and the problems of local minima and loops can be fixed) by changing the learning rate dynamically during training according to the current error trend. There are various adaptive learning rate methods which differ in the heuristic rule that alters the training rate (e.g., delta-bar-delta [52] or Rprop [99]).

The problem of especially deep networks is that the first layers often learn very slowly – because the term δ_{pj} is diminished when back-propagated through the network layers. For each neuron j, the back-propagated error δ_{pj} is a linear combination of the error terms form the above layer multiplied by the derivative of the transfer function $f'(\xi_j)$ (1.18). However, the value of $f'(\xi_j)$ is for the common transfer functions between 0 and 1 (see Figure 1.2 on page 12). Therefore, δ_{pj} may become very small after passing through several network layers and the training process may become imbalanced.

A possible solution represent local learning rates α_j, unique for each neuron j [95]. To compensate the decreasing δ_{pj}, α_j can grow in the direction from the output to the input layer. For neurons in the first layers, α_j can be even much greater than 1.

In steep valleys of the error function, following the negative gradient of the error function can lead to frequent and huge oscillations even for relatively small values of the learning rate α. To fix this problem, a common modification of the BP-algorithm – learning with momentum – can be useful. The idea is to stabilize the adaptation process by taking into account also previous weight changes. The formula 1.12 is modified in the following way:

$$\Delta w_{ij}(t+1) \;\simeq\; -\alpha \frac{\partial E_p}{\partial w_{ij}} + \alpha_m \Delta w_{ij}(t), \tag{1.24}$$

where $0 \leq \alpha_m < 1$ is the momentum rate.

When learning with momentum, the weight vector movement tends to be more stable. This may speed up training in flat regions of the error surface and

stabilize oscillations in narrow valleys. However, if the momentum rate is too high compared to the learning rate (and with respect to the solved task), the algorithm may skip shallow local minima of the error function and may not be able to follow a curved valley in the error surface. Simultaneous tuning of both the learning rate and the momentum rate is necessary, which brings a further difficulty into the training process.

If the training time is a critical option, there is also a possibility to use a more sophisticated (e.g., second order or conjugate gradient) optimization method. A disadvantage of faster training algorithms is, that their generalization ability can be comparable or even worse than for the standard BP-algorithm, because their chance of overtraining (during the same amount of time) is greater. However, various heuristics and even sophisticated techniques can be used without regard to the chosen training algorithm to assure sufficient generalization.

A common approach to avoid overtraining is the so-called early stopping (a stop criterion based on the network error on the validation data). A further possibility is to implement some of the general sampling methods (e.g., cross-validation). Ensemble learning is in a way similar to cross-validation, it is based on training several models (e.g., BP-networks with different configurations) and combining their outputs. A problem of this method is inefficiency (a large number of BP-networks needs to be trained) and no guarantee of improved generalization compared to a single well-tuned model.

Sensitivity to parameters One of the main drawbacks of the BP-algorithm is its high sensitivity to the choice of initial parameters with respect to the given problem. When applied to real-world applications, the method is sometimes accused of poor performance and poor generalization, which is however an impact of little effort invested in tuning of the parameters (and poor preprocessing of the training set).

We already discussed the issue of the learning rate α and momentum rate α_m. Further options that have to be pre-set, are the topology, transfer functions and their parameters, error function, weight initialization technique, stop conditions or weight update strategy (e.g., batch learning).

The search for an optimal combination of values of critical network parameters, such as the learning rate or topology, is a demanding part of the training process. The common approach is to train the BP-network more times with various parameters' settings and then to choose the best model. This is easy to implement, but usually extremely time consuming. Another possibility is to use methods that set the parameters dynamically (such as adaptive learning rate methods) or alternative training algorithms with less optional parameters or more robust to their actual setting (i.e., Quickprop [32]).

Maybe the most critical BP-network parameter is its topology. It should correspond to the complexity of the analyzed data. If the network is too small, it is not able to learn the task properly. On the contrary, if it is too large, it has a tendency to overtrain. Therefore, when training a larger network, we also need more training patterns to assure good generalization.

Unfortunately, even if the optimum size of the network were known, it might be difficult to train such a network completely from scratch [103], because it may

require a very specific set of weights. For larger networks there may be more ways to fit the data and therefore training may be faster and easier. Pruning techniques applied to larger already trained networks might represent a viable option in such a case [96]. An alternative to pruning methods is the opposite approach – the constructive methods (e.g., cascade correlation [33]).

Occam's razor, one of the basic principles of machine learning, prefers simple models to the complex ones. A simple model doesn't mean in the context of BP-networks just a smaller topology, but also smoother network function, lower curvature and a larger margin set along the separating hyper-planes. Such BP-networks are expected to have smaller VC-dimension and generalize better [16, 46, 121].

To reduce the model complexity, an additional prior information about the problem beside the training set may be useful – in the form of a hint function [2], or in the form of additional training patterns. Another possibility is to add an appropriate penalty term to the error function. A widely-used example of such a regularization method is weight decay [119] that forces all the network weights to be smaller.

Training with noisy data (jitter) is based on adding noise to the input patterns during training. It was empirically proved to improve generalization and smoothness for a wide range of problems especially when there are not enough training patterns [69, 104]. However, it can be quite inefficient, because for some tasks a large (and a priori unknown) number of noisy patterns need to be generated.

Transparency of internal structure Another big disadvantage of standard BP-networks is that the model doesn't tend to develop a transparent network structure. For such networks, it is not clear, what is the relation between the training data and the weights and activities of hidden neurons. Therefore it is extremely difficult to 'guess' the real meaning of every particular hidden or even input neuron for a proper network output. Extracting knowledge, especially rules, from the model is extremely difficult. Such networks often use small differences of neuron outputs to distinguish between the presented patterns.

A possible solution is a penalty term added to the error function that enforces larger margins along the separating hyper-planes or even clear internal representation of the particular hidden neurons. An example of such approach is the method of learning internal representation [86].

It is also expected that it may be easier to extract the knowledge from BP-networks with simple and smooth network functions. Therefore, pruning techniques and other methods reducing their VC-dimension may be helpful also in this respect.

Although there are various fast training algorithms and techniques that improve generalization, sometimes the standard BP-algorithm itself can yield results similar to more sophisticated approaches – especially for simpler tasks with a low dimension of the inputs and a sufficient number of training patterns. However, even in such a case, the procedure of parameter tuning can be difficult and time-consuming. For large-scale and demanding tasks (like data and web mining or multimedia data processing), the more refined methods can definitely mean a significant improvement.

2. Goals of the thesis

2.1 Introduction

Real-world data often contains non-linear dependencies, that are very difficult to extract by means of standard methods. The process of knowledge-extraction is thus often very time-consuming, without the guarantee of success. We want to find such dependencies as quickly as possible and to describe them as simply as possible.

To solve this task, we decided to use the computational model of BP-networks. Our choice has several reasons: BP-networks are a robust computational model with good approximation and generalization abilities, capable of finding non-linear dependencies in the data. Moreover, BP-networks are applicable to a wide range of complex problems. Both the model and its training process can be easily modified to overcome possible difficulties (e.g., overtraining) and achieve additional requirements (e.g., on the internal structure of the model or on the speed of the training process).

On the other hand, the BP-network model has a tendency to create relatively complex internal structures. When compared to the classical statistical models (e.g., linear regression, decision trees), BP-networks are more likely to find non-linear dependencies in the data. However it may be more difficult to describe these relations (e.g., in the form of rules).

There are more reasons to seek a robust, transparent and simple internal structure of the BP-network: The main reason is the transparency and comprehensibility of the model. Moreover, if the created network function is overtrained rather than smooth, the model tends to be sensitive to noise in the data and generalize worse.

As we discussed in Section 1.5, the prediction and generalization abilities of a BP-network depend strongly on many aspects, including an appropriate choice of the topology and other tunable parameters. A further factor that affects generalization is the quality of the training set (particularly, the relevance and mutual dependence of the input features). Excessive task dimensionality is a serious problem of many practical implementations of BP-networks.

For these reasons, there has been a strong effort in recent years to develop techniques that will force the system to find by itself the optimal topology and to ignore redundant and irrelevant inputs. Further methods intend to force the BP-network model to create a robust, transparent and simple internal structure. Other methods are focused on fast training or on generalization improvement.

Unfortunately, a large number of the techniques that solve one of the discussed problems make some of the other problems worse. For example, techniques, which effectively simplify the BP-network structure (e.g., pruning methods), are usually very costly and parameter-sensitive. To reduce the problems of the particular methods, a possible solution is to combine several methods together (e.g., a fast training algorithm that generalizes worse with a technique that avoids overtraining).

2.2 Statement of the main goals

We focus on the usage of the computational model of BP-networks for fast extraction of non-linear dependencies that are hidden in the data. For this reason,

the main goal of this work is to develop a general framework for BP-network training with the following features:

- The trained BP-network should approximate well the desired network function and generalize adequately.
- The training process should be as fast as possible.
- The applied methods should have just a small number of tunable parameters and be robust to the choice of their values.
- The training algorithm should force the BP-network model to create a robust, transparent and simple internal structure.
- The model itself should identify relevant input features in the data.
- The formed network structure should simplify knowledge extraction from the model and make the interpretation of the extracted knowledge easier.

Better results can be achieved when combining advantages of several techniques together. Therefore, to reach our goals, our framework will take advantage of several existing and newly developed techniques for BP-network training.

The goals of the work will be achieved through the following steps:

1. We will theoretically analyze and discuss the related methods and their features. Related techniques will be divided into the following categories: fast training of BP-networks, feature selection, structure optimization, generalization improvement, and creation of a transparent network structure.
2. We will design and implement the framework for BP-network training that will have the above-specified properties.
3. We will evaluate the performance of the proposed methods. The behavior and the characteristics of the framework will also be compared experimentally with alternative techniques for BP-network training, regularization, feature selection and pruning.

The experiments will be performed on several data sets of various properties – discrete and continuous, artificial and real-world, simple and complex. To achieve comprehensible and sufficiently general test results, representative datasets from the respective groups will be chosen carefully. Consequently, an appropriate preprocessing of the data will be done.

The experimental analysis will concentrate on practical features of the methods in order to answer the question whether and how well the set goals were achieved.

2.3 Chronological structure of the goals

As already stated, the main aim of this thesis is to develop and experimentally evaluate a general framework for BP-network training, with an emphasis put on the above-discussed requirements. To structure our work chronologically based on the successive progress of our research in the past years, we will now divide our goal into three sub-goals. Later in this thesis, in Chapter 4, we will propose three successive versions of our framework that will focus on solving the respective sub-goals.

1. **Fast knowledge extraction:** We seek a training algorithm that will find a transparent structure of the entire network automatically during training. At the same time, the training algorithm should be fast and it should not reduce the prediction and generalization abilities of the trained BP-network.
2. **Topology simplification:** We want the training algorithm to find an adequate and simple topology of the BP-network automatically during training. Moreover, it should identify relevant input features in the data and important hidden neurons.
3. **Fast creation of a simple and clear internal structure:** We search for an efficient training algorithm, that will fulfill both previous goals at once. Namely, it should be fast and generalize well. It should also automatically identify important input features and form a simple and clear structure of the entire network. In this way, it should facilitate knowledge extraction from the model.

In the following paragraphs, we will discuss these three sub-goals in more detail.

2.3.1 The first goal – Fast knowledge extraction

Our first goal is to design a framework for training of BP-networks, that will provide:

- **Speed:** A fast training algorithm, that doesn't have many tunable parameters and is robust to their choice.
- **Transparency:** Techniques that force the model to create a clear and transparent internal structure that will simplify knowledge extraction from the model.
- **Generalization:** Techniques that force the BP-network function to be smooth and generalize well.

A brief analysis of these three topics follows.

Speed The standard method for training BP-networks is the BP-algorithm. As we discussed in Section 1.5, drawbacks of this simple training technique are both the relatively low speed of the training process and its high sensitivity to the choice of initial parameters.

However, there are enhancements of the BP-algorithm that speed up the training (e.g. learning with momentum, early stopping, adaptive learning rate methods), or we can choose among the more sophisticated training techniques (e.g., second order or conjugate gradient optimization methods).

A problem of most of the fast training algorithms (e.g., the adaptive learning rate methods) is that they introduce many tunable parameters. Multiple parameter tuning is a very demanding task when training BP-networks. Some of the methods can even worsen the generalization ability of the model (e.g., relaxation methods or weight decay) [100]. Other methods are impractical for large-scale problems due to their great space complexity (e.g., quasi-Newton methods, Levenberg-Marquardt method [71]).

If we look for a fast training method with both adequate generalization and low space complexity, we can choose among the family of conjugate gradients methods. These techniques are also robust to noise and outliers in the data. Moreover, especially the Scaled conjugate gradients training algorithm (SCG) [75] reduces the number of parameters that have to be set.

Transparency Most of the standard methods for training of BP-networks (e.g., BP-algorithm, SCG, Levenberg-Marquardt) form models, where the role and importance of every single neuron or even whole parts of the original network remains unclear in general. Interpretation of knowledge extracted by a BP-or SCG-trained BP-network is difficult.

However, there are regularization techniques that help to make the internal structure more transparent – namely the method of learning condensed internal representation (IR-algorithm) and the method of learning unambiguous internal representation (UIR-algorithm) [86]. The objective of both methods is to obtain such a structure of the trained BP-network that would be equivalent to rules and enable thus easier knowledge extraction from the model.

An advantage of the IR- and UIR-methods is that they can be combined with all of the above-discussed training algorithms. They are also expected to improve the generalization ability of the trained BP-networks, because they favor smoother network functions and facilitate further pruning of the trained model [86].

A disadvantage of the IR- and UIR-methods is their sensitivity to the choice of the initial parameters that can affect also the prediction and generalization abilities of the final networks. The UIR-method suffers also from high time and space costs.

Generalization An adequate generalization ability of the BP-network model depends on many aspects (e.g., a careful choice of model parameters and preprocessing of the training set, choice of the training algorithm). Additional techniques are thus often needed to assure sufficient generalization and to avoid overtraining.

In this respect, we may highlight especially learning from hints [2], training with jitter [73, 93, 104], cross-validation and early stopping [95], and the methods for structure optimization. Early stopping and cross-validation are simple yet efficient mechanisms to prevent overtraining. Some of the more sophisticated methods (e.g., learning from hints, training with jitter, some of the regularization and pruning techniques) also try to reduce the VC-dimension of the final network structures and make the network function smoother. While training with jitter suffers from high time costs and is restricted to just some of the discussed training algorithms, learning from hints is a generally-usable method that can even make the training process faster.

To take advantage of the above-described methods and to overcome their limits, a possible solution is to combine them. Therefore, we decided to choose the most promising methods from each group and combine them together in order to achieve the above-stated requirements.

2.3.2 The second goal – Topology simplification

Our second goal is to enhance our framework for training of the BP-networks with **Structure optimization**:

- **Simplification of the topology:** Creation of a simple yet adequate model topology.
- **Measuring relevance:** Automatic detection of important and irrelevant parts of the BP-network.
- **Feature selection:** Sophisticated selection of relevant input features.

In the following paragraphs, we will discuss these topics in detail.

Simplification of the topology The topology of a BP-network is a very important model parameter. From our point of view, extracting knowledge from a smaller network with a simple structure is much easier. The optimal choice of topology is given by the task and it affects both the approximation and generalization abilities of the model. While most of the standard training algorithms work with a fixed topology, structure optimization methods (especially pruning algorithms, regularization techniques and network construction methods) try to find the optimal topology automatically during training.

Network construction techniques [5, 6] start with a small topology and incrementally add neurons and weights until a reasonable structure is reached. A disadvantage of these methods is that they usually require special training algorithms (e.g., genetic algorithms), the training process is usually very slow and the model has strong tendency to overtrain.

Pruning algorithms [94] start training with a large topology and then remove redundant parts of the model until a reasonable topology is achieved. The objects of pruning are usually edges, hidden neurons or input neurons.

An advantage of the pruning techniques is that they can be easily combined with most of the discussed training algorithms. Adequate pruning may improve generalization and prediction abilities of the model [34]. The main problem of the pruning methods is to set their optional parameters carefully. They are especially the relevance measure to identify redundant elements to be pruned, and the heuristic rules, that will control the pruning process.

Regularization techniques try to eliminate redundant elements (typically weights) from the network already during training. Most of such methods (e.g., weight decay [119], weight elimination [117]) add a penalty term to the error function that forces the BP-network, e.g., to decrease the absolute values of the weights. Weights smaller than a given threshold are then regarded as irrelevant and removed. A disadvantage of weight decay and similar methods is that they tend to decrease the generalization ability of the model and create smaller yet not transparent network structures.

Measuring relevance Some of the pruning techniques use a relevance measure to identify redundant elements to be pruned. They are thus able to identify significant input features and relevant hidden neurons or weights. If we knew, how the inputs impact the outputs and which inputs are more significant than the

other ones, we could understand better the internal structure of a BP-network and we could easier explain the underlying process.

Many relevance measures are based on the sensitivity analysis [123]. Sensitivity analysis tries to quantify the response of a computational model to parameter perturbations. There are more approaches to sensitivity measurements – with respect to the error function or with respect to the BP-network's outputs. Their advantage is a high accuracy. Their disadvantage is, that computing the exact sensitivity coefficients [34, 126] is relatively costly. However, there are also less precise, but efficient approximative measures (e.g., weight product [108] or optimal brain damage [62]).

Unfortunately, the sensitivity criteria alone are not capable of detecting all redundant neurons or weights, especially if the internal structure of the BP-network is too complex. Most of the sensitivity criteria assume that inputs and activities of hidden neurons are mutually independent and numerical. A possible solution may be to combine the chosen pruning technique with a suitable regularization technique that would make the relevance measurement easier.

Feature selection Because excessive task dimensionality started to be a problem of many computational models including BP-networks, feature selection has became an important part of data preprocessing.

There are many simple yet efficient model-independent feature selection techniques (e.g., filter methods based on correlation coefficients, on the information theory or on the clustering principle [40]). Their disadvantage is, that they are able to identify just linear dependencies between the input features and the outputs. Moreover, they are not able to identify redundant and mutually correlated input features.

In the case of BP-networks, feature selection can be considered as a part of structure optimization. Especially the above mentioned pruning techniques are useful in this respect. Some of them (e.g., sensitivity-based pruning methods) are able to identify also complex non-linear relationships between the input features and the outputs. Although such techniques usually perform better than filter methods, they suffer from higher computational costs.

To reach our second goal, we decided to use some the above-mentioned techniques for structure optimization. Especially the pruning techniques based on the sensitivity analysis [27, 28, 29, 34, 122, 126] seem to be advantageous, as they manage all our requirements (on structure optimization, feature selection and relevance measurement) at once. If we decide for pruning, we will have to choose a concrete relevance measure and a set of heuristic rules, that will control the pruning process. Another possibility is to use a suitable regularization technique to speed up and simplify pruning.

A further question is, which of the discussed techniques would fit most to the methods already included in our framework. Our effort is not only to adequately form during training as small BP-network topology as possible. The BP-network structure should also be comprehensible and it should simplify knowledge extraction from the model. Therefore, the chosen techniques should not work against the transparency of the model.

2.3.3 The third goal – Fast creation of a simple and clear internal structure

Our third goal will be to enhance our framework to achieve both the first and the second goals at once:

- **Generalization:** The framework should favor smooth BP-network functions that contribute to adequate generalization.
- **Speed:** The training process should be as fast as possible and robust to the choice of initial parameters.
- **Transparency and structure optimization** The BP-network should create a small, simple and transparent internal structure during training. It should also automatically identify relevant input features. The created model structure should simplify the further knowledge extraction.

Our second sub-goal didn't focus on efficiency. A problem of most of the methods for structure optimization (including pruning and regularization techniques) are high computational costs. The most promising pruning methods are based on the sensitivity analysis. Their advantage is that they are not in conflict with the methods for improved transparency, they improve generalization ability of the model, and that they support creation of a smooth network function.

To support the sensitivity-based pruning, a combination with a sensitivity-based regularization technique would be helpful. However, as the sensitivity coefficients are quite complex, we expect, that an exact sensitivity-regularization technique would also be very computationally costly. Therefore, in order to speed up the training process, we should look for approximative solutions – especially for an approximative sensitivity-based regularization technique.

3. Analyzed methods

3.1 Methods for fast training of BP-networks

The standard BP-training algorithm is based on the same strategy like most other optimization methods designed to minimize functions. The minimization is a local iterative process and the approximation is often given by the first or second order Taylor expansion of the error function (1.10).

Iterative weight adjustment begins in a starting point – configuration $\vec{w}_1$. For each point $\vec{w}_t$, determining the next point $\vec{w}_{t+1}$ involves two independent steps. First, the search direction $\vec{g}_t$ has to be determined, i.e., in what direction in the weight space we want to move in the search for a new point $\vec{w}_{t+1}$. Once the search direction has been found, the step size α_t has to be determined. The configuration $\vec{w}_t$ is then updated using the formula:

$$\vec{w}_{t+1} = \vec{w}_t + \alpha_t \, \vec{g}_t. \tag{3.1}$$

If the search direction is set to the negative gradient $\vec{g}_t = -E'(\vec{w}_t)$ and the step size to a constant (i.e., the learning rate), then the algorithm becomes the batch variant of the gradient descent based BP-algorithm. The gradient descent algorithm is based just on the linear approximation of the error function $E(\vec{w}+\vec{z}) \approx E(\vec{w}) + E'(\vec{w})^T \vec{z}$. The imprecise approximation may explain, why the convergence of the standard BP-algorithm is relatively slow, especially for large-scale real data. In theory, the asymptotic convergence rate of the BP-algorithm is linear [95] – at each step, the error is reduced by a constant factor.

As discussed in Section 1.5.2, faster training of the BP-algorithm can be achieved by learning with momentum or/and by using an adaptive learning rate method, that changes the learning rates dynamically during training (e.g., the algorithm of Silva and Almeida [105], delta-bar-delta [52] or Rprop [99]). For the method of Silva and Almeida, the adaptation rule has the form $\vec{w}_{t+1} = \vec{w}_t + \vec{\alpha}_t \, \vec{g}_t$. The step size is local (unique for each weight) and is updated after each epoch t according to the following heuristic:

$$\alpha_{t+1,i} = \begin{cases} u \, \alpha_{t,i}, & g_{t-1,i} \, g_{t,i} > 0 \\ d \, \alpha_{t,i}, & g_{t-1,i} \, g_{t,i} < 0 \\ \alpha_{t,i}, & \text{otherwise} \end{cases} \tag{3.2}$$

where i indexes the elements of $\vec{g}_t$ and $\vec{\alpha}_t$. $u > 1$ and $0 < d < 1$ are pre-set constants. The step size is increased, if the partial derivation has the same sign in the steps $t-1$ and t. It is decreased, if the partial derivation has the opposite sign in the steps $t-1$ and t.

Adaptive learning rate methods automate the process of tuning the parameter α and are usually much faster than the standard BP-algorithm with a sub-optimal choice of α. However, they are not guaranteed to be faster and generalize better than the well-tuned standard BP-algorithm. A disadvantage of most of the adaptive learning rate methods is that they need additional storage capacity (e.g., for the method of Silva and Almeida, for the vectors $\vec{g}_{t-1}$ and $\vec{\alpha}_t$). They also introduce further parameters to the model to be tuned.

On the contrary to the BP-algorithm and adaptive learning rate methods, the second-order optimization methods compute the search direction and step size more precisely. They use the quadratic approximation of the error function. Let

$E_{\vec{w}}$ denote the quadratic approximation of E in the neighborhood of the point $\vec{w}$:

$$E_{\vec{w}}(\vec{z}) = E(\vec{w}) + E'(\vec{w})^{\,T}\vec{z} + \frac{1}{2}\vec{z}^{\,T}E''(\vec{w})^{T}\vec{z}. \tag{3.3}$$

When minimizing $E_{\vec{w}}(\vec{z})$, we search for the solutions of the following linear equation:

$$E'_{\vec{w}}(\vec{z}) = E''(\vec{w})\vec{z} + E'(\vec{w}) = 0. \tag{3.4}$$

The Newton's method computes the solution directly:

$$\vec{z} = -\left(E''(\vec{w})\right)^{-1} E'(\vec{w}), \tag{3.5}$$

where $\left(E''(\vec{w})\right)^{-1}$ is the inverse Hessian matrix. The adaptation rule of the Newton method is:

$$\vec{w}_{t+1} = \vec{w}_t - \left(E''(\vec{w})\right)^{-1} E'(\vec{w}). \tag{3.6}$$

An advantage of the Newton's method is its fast convergence [95]. A problem is, that to ensure fast training and convergence to the minimum of the error function, it requires the Hessian matrix to be symmetric and positive definite, which is usually not true for the real-world tasks.

Definition 6. The real $N \times N$ matrix A is

1. symmetric, if $A^T = A$,
2. positive definite, if $\forall \vec{x} \neq 0,\ \vec{x} \in \Re^N : \vec{x}^{\,T} A\, \vec{x} > 0$.

Another problem is, that computing and storing the Hessian matrix and/or its inverse can be a difficult task. The inverse of the Hessian matrix needs to be computed in each iteration. It requires the storage $O(W)^2$, where W is the length of the weight vector, and up to $O(PW^2)$ operations to compute (3.6), where P is the number of training patterns [16]. The time and space complexity can be enormous, especially for larger network topologies.

The quasi-Newton methods (e.g., the Levenberg-Marquardt method [71]) try to overcome the problems of the Newton method by computing just an approximation of the Hessian matrix. However, the space complexity is still $O(W)^2$ [48]. On the other hand, the Levenberg-Marquardt algorithm is faster than other first- and second-order training algorithms for many tasks [42]. However, the method is sensitive to the choice of initial parameters (especially weights), to the noise in the data and it tends to overtrain [20]. Furthermore, its storage requirements make it inefficient and impractical for large-scale problems [42].

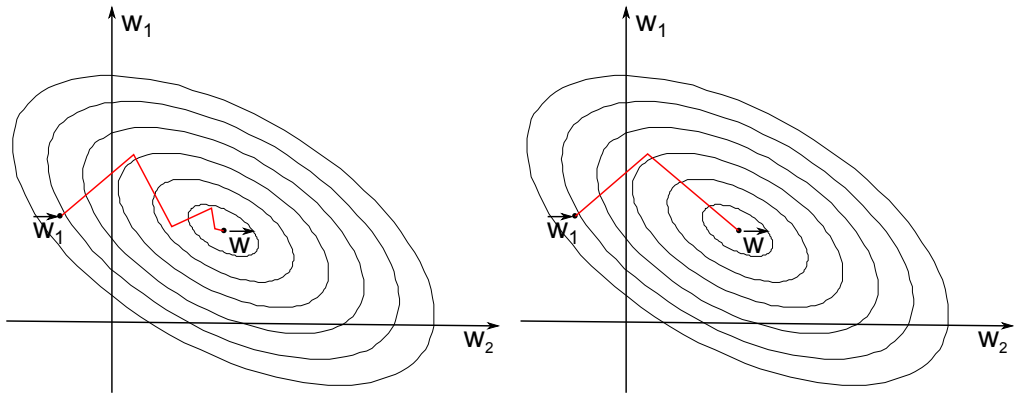
Another group of efficient training algorithms that use the second-order information are Conjugate gradients methods.

3.1.1 Conjugate gradients methods

Unlike the full second-order optimization methods, the Conjugate gradients algorithms (CG-algorithms) don't compute or store the Hessian matrix. Instead, they are based on the concept of mutually conjugate search directions.

Definition 7. The non-zero vectors $\vec{p}_1, \cdots, \vec{p}_k \in \Re^N$ are mutually conjugate with respect to the symmetric and positive definite $N \times N$ matrix M, if $\vec{p}_i^{\,T} M \vec{p}_j = 0$ for all $i \neq j,\ 1 \leq i, j \leq N$.

In each step of the algorithm, the actual search direction is determined to be conjugate (with respect to the Hessian matrix) to all preceding search directions. This prevents the respective adjustments of the weight vector to cancel out previous changes and speeds up the training process. Figure 3.1 shows a comparison of the convergence of the BP-algorithm and the CG-algorithms when minimizing a quadratic error function.



(a) BP-algorithm (b) CG-algorithms

Figure 3.1: Example of the convergence of the BP-algorithm and CG-methods when minimizing a quadratic error function. $\vec{w}_1$ is the starting configuration, while $\vec{w}$ is the final configuration.

On the contrary to the Newton and quasi-Newton methods, the space complexity of the CG-algorithms is comparable to the standard Back-propagation. They require only $O(W)$ memory usage with W being the size of the weight vector. That makes this class of models convenient also for large-scale problems with a large number of input features. Moreover, the CG-algorithms converge much faster than the BP-algorithm for many tasks [11, 21]. The training time of the efficient variants (e.g., the Scaled conjugate gradient method [75]), is usually comparable to the Levenberg-Marquardt algorithm (or even shorter for larger models [42]). Theoretically, the convergence rates are super-linear for most problems [95].

In the case of the CG-algorithms, the training process consists of independent one dimensional line-searches along the mutually conjugate directions. Similarly to other optimization methods, the adaptation rule has the form (3.1): $\vec{w}_{t+1} = \vec{w}_t + \alpha_t \vec{g}_t$, where α_t is the step size and $\vec{g}_t$ is the search direction. Also the process of computing the mutually conjugate search directions is iterative. In the first step, the search direction $\vec{g}_1$ is set equal to the gradient descent direction, $\vec{g}_1 = -E'(\vec{w}_1)$. In each step $t > 1$, the search direction $\vec{g}_t$ is determined as the combination of the previous search direction and the current gradient descent direction:

$$\vec{g}_t = -E'(\vec{w}_t) + \beta_t \vec{g}_{t-1}, \tag{3.7}$$

where $\beta_t \in \Re$ is a scaling factor.

The CG-algorithms are mostly efficient, if the error function $E(\vec{w})$ is locally quadratic with a positive definite Hessian matrix. In such a case, the minimum of $E(\vec{w})$ can be found by at most W iterations [16]. Unfortunately, as the error

function is usually not quadratic with a positive definite Hessian matrix, the number of steps is usually higher than W. There are only W mutually conjugate search directions, and therefore the methods need to restart periodically. During the restart, the actual search direction is set to the current steepest descend direction. The basic restart rule is to restart after every W iterations. The efficiency of the algorithm can be increased by the use of a more sophisticated restart method, e.g., the Powell-Beale restart rule [14, 91]. The algorithm is then restarted, if the Powell's inequality holds:

$$\mid \vec{g}_t^{\,T} \vec{g}_{t-1} \mid \geq c \| \vec{g}_t \|^2, \quad c \in (0,1). \tag{3.8}$$

The variants of the CG-methods differ in the way, how they compute the values of the parameters β_t and α_t. The most common formulae for the calculation of β_t without the explicit knowledge of the Hessian were introduced by Fletcher-Reeves [35]:

$$\beta_t = \frac{\| \vec{r}_t \|^2}{\| \vec{r}_{t-1} \|^2}, \tag{3.9}$$

and Polak-Ribiere [90]:

$$\beta_t = \frac{\vec{r}_t^{\,T}(\vec{r}_t - \vec{r}_{t-1})}{\| \vec{r}_{t-1} \|^2}, \tag{3.10}$$

where $\vec{r}_t = -E'(\vec{w}_t)$. These expressions are mutually equivalent for the quadratic error function, but can yield different results for real-world tasks.

If the error function $E(\vec{w})$ is locally quadratic with a positive definite Hessian matrix, the parameter α_t can be computed exactly. In the following paragraphs, we will derive the corresponding formula. Equally to the full second-order optimization methods, the techniques of conjugate gradients minimize the quadratic approximation of the error function $E_{\vec{w}}(\vec{z}) = E(\vec{w}) + E'(\vec{w})^{\,T}\vec{z} + \frac{1}{2}\vec{z}^{\,T}E''(\vec{w})\vec{z}$ by searching for the solution $\vec{z}^*$ of the equation (3.4): $E'_{\vec{w}}(\vec{z}) = E''(\vec{w})\vec{z} + E'(\vec{w}) = 0$.

Let $\vec{g}_1, ..., \vec{g}_W$ be mutually conjugate with respect to the positive definite Hessian matrix $E''(\vec{w})$. They form the basis in $\Re^W$ [75]. Let $\vec{w}$ be the initial weight vector and $\vec{z}_1 = \vec{0}$ be the starting point when searching for $\vec{z}^*$ minimizing $E_{\vec{w}}(\vec{z})$. Then $\vec{z}^* - \vec{z}_1$ can be expressed as the linear combination of the vectors $\vec{g}_1, \cdots, \vec{g}_W$:

$$\vec{z}^* - \vec{z}_1 = \sum_{t=1}^{W} \alpha_t \vec{g}_t \; ; \; \alpha_t \in \Re \tag{3.11}$$

If the coefficients α_t are known, $\vec{z}^*$ can be computed by $\vec{z}^* = \vec{z}_1 + \alpha_1\vec{g}_1 + \cdots + \alpha_W\vec{g}_W$, or iteratively in W steps:

$$\begin{aligned} \vec{z}_{t+1} &= \vec{z}_t + \alpha_t\vec{g}_t \; ; \; 1 \leq t \leq W, \\ \vec{z}^* &= \vec{z}_{W+1} \end{aligned} \tag{3.12}$$

A question remains, how to compute the coefficients α_t minimizing $E_{\vec{w}}(\vec{z})$. Let $\vec{w}_1 = \vec{w} + \vec{z}_1 \; ; \; \vec{w}_{t+1} = \vec{w}_t + \vec{z}_{t+1} \; ; \; 1 \leq t \leq W$. Then $\vec{w}_{W+1} = \vec{w}_1 + \vec{z}^*$ is the final weight vector. Because $\vec{g}_i^{\,T}E''(\vec{w}_t)\vec{g}_j = 0$ for all $i \neq j, \; 1 \leq i, j \leq W$, by multiplying the equation (3.11) with $\vec{g}_t^{\,T}E''(\vec{w}_t)$, we gain:

$$\vec{g}_t^{\,T}(E''(\vec{w}_t)\vec{z}^* - E''(\vec{w}_t)\vec{z}_1) = \alpha_t\vec{g}_t^{\,T}E''(\vec{w}_t)\vec{g}_t. \tag{3.13}$$

Because of (3.4), we can substitute $-E'(\vec{w}_t)$ for $E''(\vec{w}_t)\vec{z}^*$ and compute the coefficients α_t minimizing $E_{\vec{w}}(\vec{z})$:

$$\alpha_t = \frac{\vec{g}_t^T(-E'(\vec{w}_t) - E''(\vec{w}_t)\,\vec{z}_1)}{\vec{g}_t^T\,E''(\vec{w}_t)\,\vec{g}_t} = \frac{-\vec{g}_t^T\,E'_{\vec{w}_t}(\vec{z}_1)}{\vec{g}_t^T\,E''(\vec{w}_t)\,\vec{g}_t}. \tag{3.14}$$

To avoid the evaluation of the Hessian matrix, the expression (3.14) can be simplified in the following way: The value of $E''(\vec{w}_t)\,\vec{g}_t$ can be approximated by the vector $\vec{s}_t$:

$$\vec{s}_t = \frac{E'(\vec{w}_t + \sigma_t\,\vec{g}_t) - E'(\vec{w}_t)}{\sigma_t}\ ;\ 0 < \sigma_t << 1. \tag{3.15}$$

Then, the parameter α_t can be computed by:

$$\alpha_t = \frac{-\vec{g}_t^T\,E'(w_t)}{\vec{g}_t^T\,\vec{s}_t} \approx \frac{-\vec{g}_t^T\,E'_{w_t}(\vec{z}_1)}{\vec{g}_t^T\,E''(\vec{w}_t)\,\vec{g}_t}. \tag{3.16}$$

If the error function is locally quadratic and the Hessian matrix $E''(\vec{w})$ is positive definite, the process is guaranteed to find the global minimum of $E_{\vec{w}}$. Otherwise, it can easily fail. Unfortunately, the quadratic approximations on which the algorithms work can be very poor when the current point is far from the desired minimum. The solution to the problem is to alter the search in such a way that it prevents the difficulties with the indefinite Hessian matrices. Therefore the 'classical' CG-algorithms don't use the exact calculation, but find the value of the step size α_t by a line-search along the search direction $\vec{g}_t$:

$$\alpha_t = \text{argmin}_\alpha E(\vec{w}_t + \alpha\vec{g}_t) \tag{3.17}$$

using, e.g., the Charalambous' method [21]. For the quadratic error function, the Charalambous' method will find the optimal step size, otherwise it at least ensures that the error doesn't increase at any step.

Algorithm 3.1 summarizes the steps of the 'classical' CG-algorithms (with the Fletcher-Reeves expression for the calculation of β_t).

The line-search per each iteration to determine a better step size α_t is a critical part of the 'classical' CG-training, because it requires a large amount of computational time. The main reason is, that it is necessary to compute the value of the error function and the gradient several times, which can be time-expensive. A further disadvantage of this approach is that it introduces problem-dependent parameters that need to be tuned (e.g., the maximum number of iterations of the line-search). In the following section, we will discuss the method of scaled conjugate gradients that computes the parameter α_t in a different and more effective way.

3.1.2 Scaled conjugate gradients

An extremely efficient variant to the classical conjugate gradients methods represents the method of scaled conjugate gradients (SCG). This method has been proposed by Møller in [75]. It avoids the line-search, typical for other conjugate gradients methods. Instead, it computes the parameter α_t in a way inspired by the Levenberg-Marquardt approach [67]. Furthermore, SCG also eliminates most of the problem-dependent parameters, typical for the 'classical' CG-algorithms.

Algorithm 3.1 General schema of the Conjugate gradients algorithms

0. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of the size W.
   Training set $T = \{ (\vec{x}_p, \vec{d}_p) \mid \quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$.
1. *Initialization:*
   Set the discrete time variable $t = 1$.
   Initialize the weight vector $\vec{w_1}$ with small random values.
   Compute the steepest descent direction $\vec{r_1} = -E'(\vec{w}_1)$.
   Set the search direction $\vec{g}_1 = \vec{r}_1$.
2. Compute the step size α_t:
   $\alpha_t = \text{argmin}_\alpha E(\vec{w}_t + \alpha\, \vec{g}_t)$.
3. Update the weight vector and compute the new steepest descent direction:
   $\vec{w}_{t+1} = \vec{w}_t + \alpha_t \vec{g}_t$,
   $\vec{r}_{t+1} = -E'(\vec{w}_{t+1})$.
4. If the restart condition is satisfied (e.g., $t \; mod \; W = 0$),
   restart the algorithm and re-initialize the search direction $\vec{g}_{t+1}$:
   $\vec{g}_{t+1} = \vec{r}_{t+1}$.
   Else compute β_{t+1} and the new search direction:
   $\beta_{t+1} = \frac{\|\vec{r}_{t+1}\|^2}{\|\vec{r}_t\|^2}$,
   $\vec{g}_{t+1} = \vec{r}_{t+1} + \beta_{t+1}\vec{g}_t$.
5. If the stop condition is not satisfied (e.g., $\vec{r}_t \neq 0$), set $t = t + 1$
   and go to step 2.,
   else terminate and return the weight vector $\vec{w}_t$ as the desired minimum
   of the error function E.

The basic idea is to regulate the indefiniteness of the Hessian matrix by a control parameter $\lambda_t > 0$ (Lagrange multiplier). The Hessian matrix $E''(\vec{w}_t)$ is then replaced by the matrix $E''(\vec{w}_t) + \lambda_t I$. The quadratic approximation $E_{\vec{w}_t}$ of the error E in the neighborhood of the point $\vec{w}_t$ is altered in the following way:

$$E_{\vec{w}_t,\lambda_t}(\vec{z}) = E(\vec{w}_t) + E'(\vec{w}_t)^T \vec{z} + \frac{1}{2}\vec{z}^T(E''(\vec{w}_t) + \lambda\, I)\, \vec{z}. \tag{3.18}$$

The optimal step size α_t (see Eq. 3.14) can be expressed using the following expression:

$$\alpha_t = \frac{-\vec{g}_t^T E'_{\vec{w}_t,\lambda_t}(\vec{z}_1)}{\vec{g}_t^T E''(\vec{w}_t)\vec{g}_t + \lambda_t \parallel \vec{g}_t \parallel^2} = \frac{-\vec{g}_t^T E'_{\vec{w}_t,\lambda_t}(\vec{z}_1)}{\vec{g}_t^T (E''(\vec{w}_t) + \lambda_t\, I)\, \vec{g}_t} \tag{3.19}$$

and it can be controlled by the value of λ_t: the higher is the value of λ_t, the smaller is the step size. Similarly to the 'classical' CG-algorithms, the Hessian matrix is not evaluated. The value of $(E''(\vec{w}_t) + \lambda_t\, I)\, \vec{g}_t$ is approximated by the vector $\vec{s}_t$:

$$\vec{s}_t = \frac{E'(\vec{w}_t + \sigma_t\, \vec{g}_t) - E'(\vec{w}_t)}{\sigma_t} + \lambda_t\, \vec{g}_t \; ; \; 0 < \sigma_t << 1. \tag{3.20}$$

Furthermore (because $\vec{z}_1 = \vec{0}$):

$$E'_{\vec{w}_t,\lambda_t}(\vec{z}_1) = (E''(\vec{w}_t) + \lambda_t\, I)\, \vec{z}_1 + E'(\vec{w}_t) = E'(\vec{w}_t) \tag{3.21}$$

The step size α_t is computed by:

$$\alpha_t = \frac{-\vec{g_t}^T E'(\vec{w_t})}{\vec{g_t}^T \vec{s_t}} = \frac{\mu_t}{\delta_t} \approx \frac{-\vec{g_t}^T E'_{\vec{w_t},\lambda_t}(\vec{z}_1)}{\vec{g_t}^T (E''(\vec{w_t}) + \lambda_t I)\, \vec{g_t}}, \tag{3.22}$$

where

$$\delta_t = \vec{g_t}^T \vec{s_t} \quad , \quad \mu_t = -\vec{g_t}^T E'(\vec{w_t}). \tag{3.23}$$

The value of the parameter $\delta_t \approx \vec{g_t}^T (E''(\vec{w_t}) + \lambda_t I)\, \vec{g_t}$ indicates, whether the altered Hessian matrix is positive definite: If for a given time step t : $\delta_t \leq 0$, the altered Hessian matrix is not positive definite. In such a case, the value of λ_t should be increased in order to make δ_t positive and the altered Hessian matrix positive definite. Let $\overline{\lambda_t}$, $\overline{\vec{s_t}}$ a $\overline{\delta_t}$ denote the new values of λ_t, $\vec{s_t}$ a δ_t, respectively. The values of $\overline{\vec{s_t}}$ and $\overline{\delta_t}$ can be computed using the following formulae:

$$\begin{aligned} \overline{\vec{s_t}} &= \frac{E'(\vec{w_t} + \sigma_t \vec{g_t}) - E'(\vec{w_t})}{\sigma_t} + \overline{\lambda_t}\, \vec{g_t} \\ &= \frac{E'(\vec{w_t} + \sigma_t \vec{g_t}) - E'(\vec{w_t})}{\sigma_t} + \lambda_t \vec{g_t} + (\overline{\lambda_t} - \lambda_t)\, \vec{g_t} \\ &= \vec{s_t} + (\overline{\lambda_t} - \lambda_t)\, \vec{g_t}\,, \end{aligned} \tag{3.24}$$

$$\begin{aligned} \overline{\delta_t} &= \vec{g_t}^T \overline{\vec{s_t}} = \vec{g_t}^T \vec{s_t} + \vec{g_t}^T (\overline{\lambda_t} - \lambda_t)\, \vec{g_t} \\ &= \delta_t + (\overline{\lambda_t} - \lambda_t)\ \| \vec{g_t} \|^2\,. \end{aligned} \tag{3.25}$$

To ensure $\overline{\delta_t} > 0$, it is necessary that $\overline{\lambda_t} > \lambda_t - \frac{\delta_t}{\|\vec{g_t}\|^2}$.

A question is, how to set $\overline{\lambda_t}$ to get an optimal solution, i.e., to force $E_{\vec{w_t},\lambda_t}$ to be a good quadratic approximation of E in the neighborhood of $\vec{w_t}$. Møller in [75] recommends to set

$$\overline{\lambda_t} = 2\,(\lambda_t - \frac{\delta_t}{\| \vec{g_t} \|^2}). \tag{3.26}$$

In such a case:

$$\begin{aligned} \overline{\delta_t} &= \delta_t + (\overline{\lambda_t} - \lambda_t)\ \| \vec{g_t} \|^2 \\ &= \delta_t + (2\,\lambda_t - 2\,\frac{\delta_t}{\| \vec{g_t} \|^2} - \lambda_t)\ \| \vec{g_t} \|^2 \\ &= -\delta_t + \lambda_t\ \| \vec{g_t} \|^2, \end{aligned} \tag{3.27}$$

and

$$\begin{aligned} \overline{\vec{s_t}} &= \vec{s_t} + (\overline{\lambda_t} - \lambda_t)\, \vec{g_t} = \vec{s_t} + (2\,\lambda_t - 2\,\frac{\delta_t}{\| \vec{g_t} \|^2} - \lambda_t)\ \vec{g_t} \\ &= \vec{s_t} + \left(\lambda_t - 2\frac{\delta_t}{\| \vec{g_t} \|^2}\right) \vec{g_t}\,. \end{aligned} \tag{3.28}$$

Because $\lambda_t > 0, \delta_t \leq 0$, it is ensured that $\overline{\delta_t} = -\delta_t + \lambda_t\ \| \vec{g_t} \|^2 > 0$

Because the parameter λ_t alters the Hessian matrix in an artificial way, $E_{\vec{w_t},\lambda_t}$ may not be a very good approximation of E in some points even if the altered Hessian matrix is positive definite [75]. A solution is to measure the quality of the quadratic approximation $E_{\vec{w_t},\lambda_t}$ and dynamically change λ_t in the following way:

If $E_{\vec{w}_t,\lambda_t}(\alpha_t\vec{g}_t)$ is close to $E(\vec{w}_t + \alpha_t\vec{g}_t)$, λ_t should be reduced. If the quadratic approximation is poor, λ_t should be increased. This can be achieved by the comparison parameter Δ_t, that measures the quality of the quadratic approximation:

$$\begin{aligned}\Delta_t &\approx \frac{E(\vec{w}_t) - E(\vec{w}_t + \alpha_t\vec{g}_t)}{E(\vec{w}_t) - E_{\vec{w}_t,\lambda_t}(\alpha_t\,\vec{g}_t)} = \frac{E(\vec{w}_t) - E(\vec{w}_t + \alpha_t\,\vec{g}_t)}{-E'(\vec{w}_t)^T\,\alpha_t\vec{g}_t - \frac{1}{2}\alpha_t\vec{g}_t^T\,(E''(\vec{w}_t) + \lambda_t\,I)\,\alpha_t\vec{g}_t} \\ &= \frac{E(\vec{w}_t) - E(\vec{w}_t + \alpha_t\,\vec{g}_t)}{\alpha_t\mu_t - \frac{1}{2}\alpha_t^2\delta_t} = \frac{2\,\delta_t\,[E(\vec{w}_t) - E(\vec{w}_t + \alpha_t\,\vec{g}_t)]}{\mu_t^2}.\end{aligned} \tag{3.29}$$

The parameter λ_t is altered in each step t of the SCG-algorithm using the following rules:

- If $\Delta_t \geq 0.75$, then $\lambda_{t+1} = \frac{1}{4}\,\lambda_t$.
- If $\Delta_t < 0.25$, then $\lambda_{t+1} = \lambda_t + \frac{\delta_t\,(1-\Delta_t)}{\|\vec{g}_t\|^2}$.

Algorithm 3.2 describes in detail the respective steps of the SCG-algorithm.

The SCG-algorithm belongs to the most efficient methods for BP-network training. The calculation complexity of the SCG-algorithm is $O(W^2)$ per iteration and just O(W) memory usage [75]. For many tasks, the convergence is a degree faster then for the BP-algorithm, while the approximation and generalization abilities of the trained networks are comparable [42], [95]. It is not clear, which variant of the CG-algorithms is the best in practice, because different variants outperform the others for different tasks [42]. On the contrary to the classical CG-algorithms, the SCG-algorithm computes the step size in a more sophisticated way. Moreover, it minimizes the number of tunable parameters. For this reason, we preferred to use this method in our framework.

Algorithm 3.2 Scaled conjugate gradients algorithm (SCG):

0. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of the size W.
   Training set $T = \{ (\vec{x}_p, \vec{d}_p) \mid \quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$,
1. *Initialization:*
   Set the discrete time variable $t = 1$ and set $success = true$.
   Initialize the weight vector $\vec{w_1}$ with small random values.
   Set scalars such that $0 < \sigma \leq 10^{-4}$, $0 < \lambda_1 \leq 10^{-8}$ and $\overline{\lambda}_1 = 0$.
   Calculate the steepest descent direction $\vec{r}_1 = -E'(\vec{w}_1)$.
   Set the search direction $\vec{g}_1 = \vec{r}_1$.
2. If $success = true$, then calculate the second-order information:
   $$\sigma_t = \frac{\sigma}{\| \vec{g}_t \|} \quad ; \quad \vec{s}_t = \frac{E'(\vec{w}_t + \sigma_t \vec{g}_t) - E'(\vec{w}_t)}{\sigma_t} \quad ; \quad \delta_t = \vec{g}_t^T \vec{s}_t \quad .$$
3. Scale $\vec{s}_t$, δ_t:
   $$\vec{s}_t = \vec{s}_t + (\lambda_t - \overline{\lambda_t})\, \vec{g}_t \quad ; \quad \delta_t = \delta_t + (\lambda_t - \overline{\lambda_t}) \parallel \vec{g}_t \parallel^2 \quad .$$
4. If $\delta_t \leq 0$, then make the Hessian matrix positive definite by setting:
   $$\vec{s}_t = \vec{s}_t + \left(\lambda_t - 2\frac{\delta_t}{\| \vec{g}_t \|^2}\right) \vec{g}_t \quad ; \quad \overline{\lambda}_t = 2\left(\lambda_t - \frac{\delta_t}{\| \vec{g}_t \|^2}\right) \quad ;$$
   $$\delta_t = -\delta_t + \lambda_t \parallel \vec{g}_t \parallel^2 \quad ; \quad \lambda_t = \overline{\lambda_t} \quad .$$
5. Calculate step size α_t : $\quad \mu_t = \vec{g}_t^T \vec{r}_t \quad ; \quad \alpha_t = \frac{\mu_t}{\delta_t} \quad .$
6. Calculate the comparison parameter Δ_t :
   $$\Delta_t = \frac{2\,\delta_t\,[E(\vec{w}_t) - E(\vec{w}_t + \alpha_t \vec{g}_t)]}{\mu_t^2} \quad .$$
7. If $\Delta_t \geq 0$, then a successful reduction in the value of the error function E can be made:
   $$\vec{w}_{t+1} = \vec{w}_t + \alpha_t \vec{g}_t \quad ; \quad \vec{r}_{t+1} = -E'(\vec{w}_{t+1}) \quad ;$$
   $$\overline{\lambda}_t = 0 \quad ; \quad success = true \quad .$$
   7.a If $t \bmod W = 0$, then restart the algorithm by setting: $\vec{g}_{t+1} = \vec{r}_{t+1}$,
   else create a new conjugate direction:
   $$\beta_{t+1} = \frac{\vec{r}_{t+1}^T (\vec{r}_{t+1} - \vec{r}_t)}{\| \vec{r}_t \|^2} \quad \text{and} \quad \vec{g}_{t+1} = \vec{r}_{t+1} + \beta_{t+1} \vec{g}_t \quad .$$
   7.b If $\Delta_t > 0.75$ then reduce the scale parameter: $\lambda_t = \frac{1}{4} \lambda_t$,
   else reduction in error is not possible:
   $$\overline{\lambda}_t = \lambda_t \quad ; \quad success = false \quad .$$
8. If $\Delta_t < 0.25$ then increase the scale parameter: $\lambda_t = \lambda_t + \frac{\delta_t (1 - \Delta_t)}{\| \vec{g}_t \|^2}$
9. If the stop condition is not satisfied (e.g., the steepest descent direction $\vec{r}_t \neq 0$), set $t = t + 1$ and go to step 2),
   else terminate and return the weight vector $\vec{w}_t$ as the desired minimum for the error function E.

3.2 Feature selection techniques

When using the BP-networks and other computational models (e.g., decision trees) to solve real-world problems, the proper preparation of the training data is essential. That involves particularly the process of feature subset selection (FSS) that focuses on the serious problem of excessive task dimensionality [114]. The input to the FSS process is a set on input features that includes relevant features important for the training but also redundant and irrelevant features that will make the training more difficult or even impossible. The aim of the FSS techniques is to identify a subset of significant input features and discard the remaining ones.

There are several reasons, why to do feature selection [41]:

1. General data reduction decreases storage requirements and it may make the training process faster.
2. When reducing the feature set, we save resources in the next round of data collection and preprocessing.
3. A smaller set of informative and relevant features may improve the model performance and increase its predictive accuracy. It is useful especially if there is only a limited number of training patterns available.
4. Feature selection may supply to better data understanding and improve the comprehensibility of the model.

In practice, superfluous inputs may also lead to worse generalization. Intuitively, the intrinsic generalization of an input pattern $\vec{x}_p$ includes all the input patterns $\vec{x}_q, q \neq p$ that cannot be distinguished from $\vec{x}_p$ under the (low-dimensional) internal representation r: $r(\vec{x}_p) = r(\vec{x}_q)$. Quite naturally, one would like to group all the patterns from each class into a small number of equivalence classes, with each class having its cardinality as large as possible. A lower number of equivalence classes implies reduced dimensionality of trained networks resulting into a lower VC-dimension and improved generalization [114]. This general idea has inspired several cluster-based feature selection techniques and methods based on the sensitivity analysis.

Some of the techniques for feature extraction like PCA [53] try to identify mutual correlations among the input features. Most of such techniques (e.g., PCA) can detect just linear dependencies among the data. However, some sophisticated methods can also detect non-linear relations among the data, e.g., the sensitivity-based methods [29, 34].

Generally, feature selection methods can be classified into three categories: filter, wrapper and embedded methods [17]. Filter methods are independent of the selected computational model and training algorithm, which may be very complex and sophisticated. Instead, they focus on efficiency and they are based on a simple, often heuristic principle. Wrapper models use a predictor as a black box to score the feature subsets. The embedded methods are usually specific to the chosen computational model, while the feature selection process is a part of the model training. Typically, wrapper and embedded methods have a better performance than filter methods but require heavier computation costs. In the following paragraphs, we will describe some of the methods for feature selection in more detail.

3.2.1 Feature ranking methods

A high percentage of the filter methods are based on feature ranking. They use a relevance measure that computes a complete order of features, while each input feature is processed separately. Let R_i denote the relevance of the input feature i , $i = 1, ..., n$, where n is the total number of input features. Then the FSS process consists of the following steps:

1. Compute the relevance coefficients R_i for each input feature $i = 1, ..., n$.
2. Rank the input features according to their relevance.
3. Select n' features with the highest values of relevance.

Correlation coefficients and information theoretic relevance measures

To detect relevant input features, we can choose among various simple or sophisticated relevance measures [40, 41]. A simple approach to feature relevance measurement represent the correlation coefficients. They assess the degree of linear dependence of individual input feature with the outputs. A widely used correlation coefficient is the Pearson's coefficient, that is restricted to tasks with just one numerical output. For the wanted model function $\varphi : \Re^n \to \Re$ and the training set $T = \{[\,\vec{x}_p, y_p\,] \,|\quad \vec{x}_p \in \Re^n, \quad y_p \in \Re, \quad p \in \{1, \ldots, P\}\}$, the Pearson's coefficient is defined as

$$R_i = \frac{cov(X_i, Y)}{\sqrt{var(X_i)var(Y)}}\,, \tag{3.30}$$

with the estimate:

$$R_i = \frac{\sum_{p=1}^{P}(x_{pi} - \bar{x}_i)(y_p - \bar{y})}{\sqrt{\sum_{p=1}^{P}(x_{pi} - \bar{x}_i)^2 \sum_{p=1}^{P}(y_p - \bar{y})^2}}, \tag{3.31}$$

where $\bar{y} = \text{mean}_{\{q\in\{1,\ldots,P\}\}}\, y_q$ and $\bar{x}_i = \text{mean}_{\{q\in\{1,\ldots,P\}\}}\, x_{qi}$, X_i is the i-th input variable and Y is the output variable. Further correlation coefficients are based on the classical test statistics (e.g.,T-test, F-test, χ^2-test). Disadvantage of the correlation criteria such as Pearson's coefficient is that they can only detect linear dependencies between input features and the output.

Several filters are based on Information theory [40]. Many of such filters rely on empirical estimates of the mutual information between each input feature and the output:

$$R_i = MI(Y, X_i) = \int_{x_i}\int_{y} p(x_i, y) log_2 \frac{p(x_i, y)}{p(x_i)p(y)} dx_i dy = H(Y) - H(Y|X_i), \tag{3.32}$$

where $p(x_i)$ and $p(y)$ are the probability densities of x_i and y, $p(x_i, y)$ is the joint density. $H(Y) = -\int_y p(y) log_2 p(y) dy$ is the entropy of Y, $H(Y|X_i) = \int_{x_i} p(x_i)(-\int_y p(y|x_i) log_2 p(y|x_i)) dx_i$ is the conditional entropy of Y given X_i. The difficulty of the information theoretic criteria is that the densities $p(x_i)$, $p(y)$ and $p(x_i, y)$ are all unknown and are hard to estimate from data, especially for the continuous data and when the number of training patterns is small.

An advantage of the simple feature ranking methods sketched above is that they are very efficient. Therefore, they can be used as preprocessing for more

sophisticated methods to spare computational costs, especially for tasks with very large numbers of features.

On the other side, most of these methods require probabilities that are not easy to estimate for continuous features, especially when the number of training patterns is small. Some of the methods are even restricted to a particular type of the data (e.g., discrete, with just one output).

A further disadvantage is, that most of the methods identify just linear dependencies between the input features and the outputs. They are also not able to identify redundant and mutually correlated input features [40]. The methods assume that the input features are mutually independent and they rank variables just based on their individual predictive qualities. However, input features that are useless by themselves can be useful together with others.

Feature construction methods

A special class of filter methods is based on the construction of higher order features from the original ones. The higher order features are then ordered based on the variance they explain and only the best features are selected. The classical representative of this approach is the Principal component analysis (PCA) [53]. This method generates linear combinations of features. The new feature vectors are orthogonal in the original space. PCA has been used successfully in many tasks to reduce the dimensionality of the data [53]. Yet in principle, PCA can detect just linear dependencies among the data.

Cluster-based relevance measures

Many of the relevance measures are restricted to classification tasks and are based on clustering [18], [24], [68]. Clustered data provide namely automatically an intrinsic equivalence class structure expected to yield improved generalization [114]. The aim of a classification task is to divide the training patterns into several classes. In such a case, the training set has the form

$$T = \{[\,\vec{x}_p, d_p\,] \mid \quad \vec{x}_p \in \Re^n, \quad d_p \in \{1, ..., cl\}, \quad p \in \{1, \ldots, P\}\}, \tag{3.33}$$

where cl is the number of classes.

The training set T can be divided by a chosen clustering method (e.g., c-means [70]) into k clusters $C_1, ..., C_k$ for a chosen $k > 0$. Let $\vec{c}_1, ..., \vec{c}_k$ be the centroids of the clusters $C_1, ..., C_k$. Let $O_l \in \{1, ..., cl\}$ be the majority class for the input patterns in cluster C_l and vector $\vec{A}_l \in [0, 1]^{cl}$, $A_{lj} = 1 \iff O_l = j$. For each training input pattern $\vec{x}_p$, $p \in \{1, ..., P\}$ belonging to the cluster C_{l_p}, let $\vec{a}_p = \vec{A}_{l_p}$. Then $\hat{T} = \{(\vec{x}_1, \vec{a}_1), ..., (\vec{x}_P, \vec{a}_P)\}$ is the new training set.

Several relevance measures may be used to select the subset $S_F \subset \{1, ..., n\}$ of relevant input features. At first, the relevance R_{li} of the feature i for the cluster C_l is computed for each input feature and each cluster:

$$R_{li} = \underset{\{p, \vec{x}_p \in C_l\}}{\text{mean}} r_{pi}, \tag{3.34}$$

where r_{pi} denotes for an input pattern $\vec{x}_p$ and feature i one of the relevance measures listed below. Then, S_F is selected in the following way: For each cluster C_l, order the features according to R_{li} in the descending order and select

the first k_l features before a great fall in R_{li}. If there occur more possible splits, that one yielding less features is chosen. The resulting feature set is the union of the feature sets for the particular clusters.

Distance-relevance [83] ***(dist)*** For the input pattern $\vec{x}_p$ belonging to the cluster C_{k_p} with the centroid $\vec{c}_{k_p}$ and the input feature i:

$$r_{pi} = -\frac{|x_{pi} - c_{k_p i}|}{\sum_{l \neq k_p} |x_{pi} - c_{li}|}. \tag{3.35}$$

Minimum-relevance [83] ***(min)*** For the input pattern $\vec{x}_p$ belonging to the cluster C_{k_p} with the centroid $\vec{c}_{k_p}$ and the input feature i:

$$r_{pi} = -\min\left(1, \frac{|x_{pi} - c_{k_p i}|}{\min_{l \neq k_p} |x_{pi} - c_{li}|}\right) \tag{3.36}$$

for $\min_{l \neq k_p} |x_{pi} - c_{li}| \neq 0$. Otherwise, $r_{pi} = -1$.

Maximum-relevance [83] ***(max)*** For the input pattern $\vec{x}_p$ belonging to the cluster C_{k_p} with the centroid $\vec{c}_{k_p}$ and the input feature i:

$$r_{pi} = -\min(1, \frac{\max_{l \neq k_p}(x_{pi} - c_{li})^{-2}}{(x_{pi} - c_{k_p i})^{-2}}) \tag{3.37}$$

for $\max_{l \neq k_p}(x_{pi} - c_{li})^{-2} \neq \infty$. Otherwise, $r_{pi} = -1$.

Entropy-relevance [55] ***(entro)*** The relevance measure R_i does not depend on clustering. For input patterns $\vec{x}_p, \vec{x}_q$, let

$$D_{pq}(S_F) = \sqrt{\sum_{i \in S_F} \left(\frac{x_{pi} - x_{qi}}{\max_r x_{ri} - \min_r x_{ri}}\right)^2}, \tag{3.38}$$

$$\hat{D}_{pq}(S_F) = 2^{\frac{D_{pq}(S_F)}{\text{mean}_{p,q}\, D_{pq}(S_F)}}, \tag{3.39}$$

$$\hat{R}(F) = -\sum_{p=1}^{n-1} \sum_{q=p+1}^{n} \Big[\hat{D}_{pq}(S_F)\, \log \hat{D}_{pq}(S_F) + \tag{3.40}$$

$$+ (1 - \hat{D}_{pq}(S_F))\, \log(1 - \hat{D}_{pq}(S_F)) \Big], \tag{3.41}$$

for the set of features $S_F \subset \{1, ..., n\}$. For the feature i, the relevance measure is:

$$R_i = \hat{R}(\{1, ..., n\}) - \hat{R}(\{1, ..., n\} \setminus i). \tag{3.42}$$

Example

We will illustrate the cluster-based approach to feature selection on the Iris data set [8] (Figure 3.2). The data set contains 150 training patterns, 4 input features and 1 output feature. The output feature indicates one of three classes ('Iris Setosa','Iris Versicolor', and 'Iris Virginica'), where each class refers to a type of iris plant. The four input features are denoted as 'Sepal length in cm' (SL), 'Sepal width in cm' (SW), 'Petal length in cm' (PL), and 'Petal width in cm' (PW).

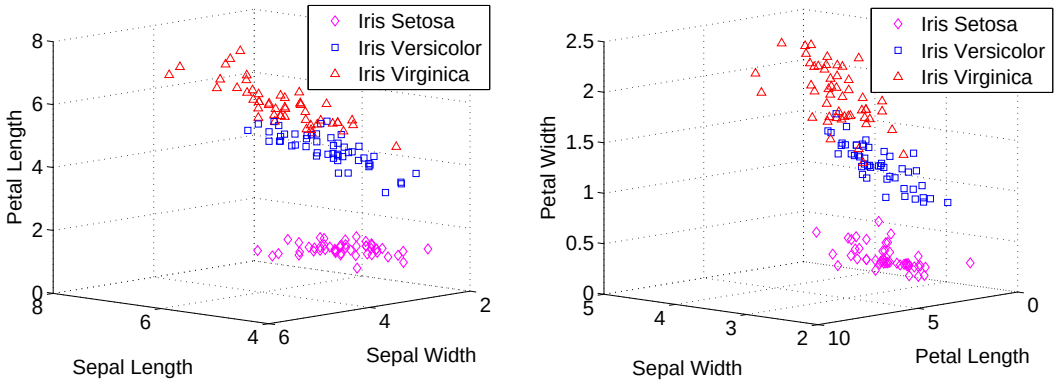


Figure 3.2: Graph of the Iris data set. The training patterns are denoted by signs that indicate one of the three classes 'Iris Setosa','Iris Versicolor', and 'Iris Virginica'.
To make the graphs comprehensible, each graph shows just three of the four input features (via axes) – 'Sepal Length', 'Sepal Width' and 'Petal Length' (on the left) and 'Sepal Length', 'Petal Length', and 'Petal Width' (on the right).

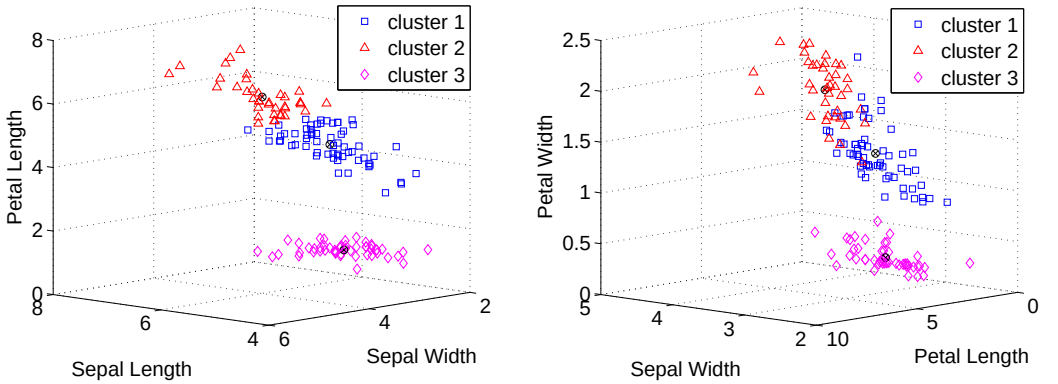


Figure 3.3: Clustering of the Iris data set into three clusters using the c-means algorithm. Centroids of the clusters are denoted by small black circles.
To make the graphs comprehensible, each graph shows just three of the four input features (via axes) – 'Sepal Length', 'Sepal Width' and 'Petal Length' (on the left) and 'Sepal Length', 'Petal Length', and 'Petal Width' (on the right).

Figure 3.3 shows a clustering of the Iris data into three clusters using the c-means algorithm. Based on the data and its clustering, the above-listed relevance measures can be computed in order to evaluate the relevance R_{li} of the particular input features $i \in \{SL, SW, PL, PW\}$ for each cluster $l \in \{1, 2, 3\}$. Bars in Figure 3.4 illustrate, how the relevances R_{li} look like for the distance-relevance and minimum-relevance measures.

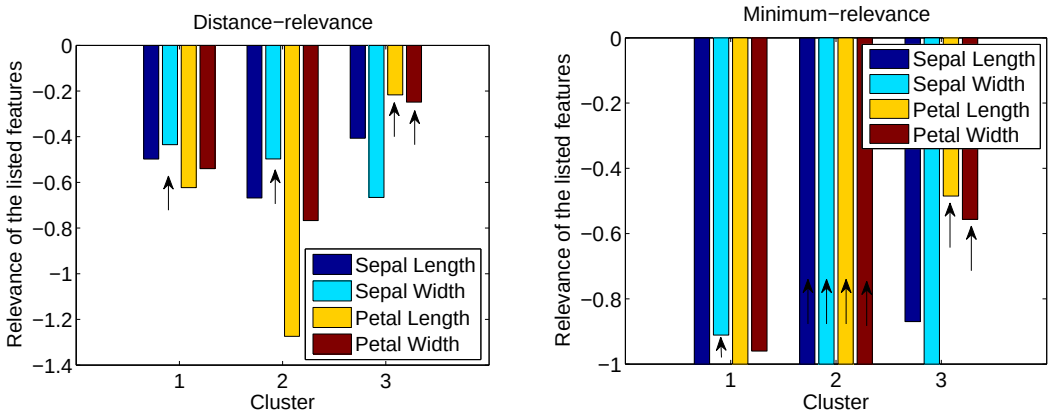


Figure 3.4: Relevance of the input features ('Sepal Length', 'Sepal Width', 'Petal Length', and 'Petal Width') for each cluster $l \in \{1, 2, 3\}$. The relevance is computed using the distance-relevance measure (on the left) and the minimum-relevance measure (on the right). The most relevant input features are denoted by black arrows.

After computing the relevance R_{li}, the most relevant features can be selected using the following heuristic: For each cluster l, order the input features according to R_{li} in the descending order and select the first k_l mostly-relevant features before a great fall in R_{li}. The resulting feature set is the union of the feature sets for the particular clusters.

Table 3.1 shows features selected using the the distance-relevance (*dist*), minimum-relevance (*min*), maximum-relevance (*max*), entropy-relevance (*entro*), Pearson's correlation coefficient (*pearson*, see Subsection 3.2.1 on page 46) and sensitivity-relevance (*sens*, see Subsection 3.2.3 on page 52) measures.

Table 3.1: Features denoted as relevant for the respective clusters and for the entire data using the chosen feature selection techniques.

cluster	*dist*	*min*	*max*	*entro*	*pearson*	*sens*
1	SW	SW	SW	SW, SL	*PL, PW*	PL, PW
2	SW	all	all	SW, PL, PW	*PL, PW*	PL, PW
3	PL, PW	PL, PW	PL, PW	PL	–	PL, PW
all	SW, PL, PW	all	all	all	SL, PL, PW	PL, PW

An interesting observation is, that for clusters 1 and 3, all of the cluster-based methods denoted almost the same features as the most relevant (i.e., SW for cluster 1 and PL, PW for cluster 3). However, the particular methods have various outcomes for cluster 2. In sum, for a majority of the cluster-based methods, each of the four input features was denoted as relevant for at least one of the three

clusters. Therefore, such methods (i.e., *min*, *max* and *entro*) selected all the input features.

Another interesting observation is, that the *sens* method (that is independent on the chosen clustering) selected the features PL, PW as the most relevant for all the clusters and also for the entire data and the respective classes ('Iris Setosa','Iris Versicolor', and 'Iris Virginica').

3.2.2 Wrapper methods

Contrary to the feature ranking methods, the wrapper methods don't measure the relevance of each input feature separately, but they assess the relative usefulness of the particular subsets of input features.

The FSS process consists in searching for an optimal subset of input features in the space of all possible feature subsets. The quality of the subsets $S_F \subset \{1, ..., n\}$ is evaluated based on the performance of the chosen computational model on S_F. In practice, important questions are:

1. Which computational model to use?
2. How to score the feature subsets based on the performance of the chosen computational model?
3. How to search the space of all possible feature subsets?

Computational models widely used in this respect include decision trees, naive Bayes, support vector machines and BP-networks, where especially BP-networks are able to identify also non-linear dependencies in the data. Performance assessments are usually done using a validation set.

An exhaustive search of the space of all possible feature subsets becomes for higher numbers of input features computationally intractable. Therefore, the wrapper methods usually use heuristic search strategies, e.g., branch-and-bound, simulated annealing, genetic algorithms, or greedy search [58]. Computationally advantageous are especially the greedy search strategies, namely forward selection and backward elimination. In forward selection, we start with a single input feature and progressively incorporate other features into larger and larger subset. In backward elimination we start with the set of all input features and progressively eliminate the least promising ones.

When compared to filter methods, wrappers are still a relatively simple, however universal approach to FSS. They are in principle applicable to any type of data. A disadvantage of wrapper methods are high computational costs, because the chosen computational model has to be trained from scratch and evaluated several times for various feature subsets. To decrease the computational complexity, there is a need to reduce the number of examined feature subsets as much as possible. However, if we choose a too rough search strategy, the method will tend to overtrain [98].

A further disadvantage of the embedded methods is, that they may yield very different feature subsets if the training data changes a little (e.g., if it is corrupted by noise), or for various initial settings of the chosen computational model [41]. A possible solution of this problem is the use of ensemble models [112]. Ensemble models combine outputs from multiple computational models to get a more precise and stable prediction. However, the price for the robustness are greater computational costs.

3.2.3 Embedded models

Ideally, the system should learn to ignore redundant and irrelevant inputs by itself. Embedded methods thus incorporate feature selection as a part of the training process. They may be faster than the wrapper methods, because the chosen computational model is trained only once. However the training process may be more complex and computationally expensive than without the embedded feature extraction.

BP-networks and feature selection

For BP-networks, the issue of feature selection can be looked at as the part of structure optimization (see Section 3.3). Especially the pruning techniques try to identify and eliminate irrelevant parts from the BP-network – edges, hidden neurons and also input neurons, that correspond to the input features.

Most of the methods for pruning of the input neurons are based on the main principles of sensitivity analysis (see Section 3.3.3 for details). A representative of this approach is the method of Fidalgo [34] that is capable of detecting also non-linear dependencies among the data. The sensitivity coefficients S_{ij} are computed as the mean absolute value of the derivatives of the j-th output with respect to the i-th input over all of the P input patterns:

$$S_{ij} = \frac{1}{P} \sum_{p=1}^{P} \left| \frac{\partial y_{pj}}{\partial x_{pi}} \right| \tag{3.43}$$

and express how and how much the solution to a given problem depends on the data. Input neurons with low sensitivity coefficients are considered to be less important and can be pruned from the network.

To measure also the curvature, both the first- and second-order partial derivatives of the output variables with respect to the input variables can be used [122].

Sensitivity relevance *(sens)* The sensitivity coefficients can be used also as a relevance measure for a feature ranking method. A neural network is then trained on the training set T using the selected training algorithm, e.g., the SCGIR algorithm (Algorithm 4.1). For the j-th BP-network output and input feature i, let S_{ij} be the sensitivity of the output j on the input i. Then

$$R_i = \operatorname*{mean}_j S_{ij}. \tag{3.44}$$

A problem of this method is that the sensitivity coefficients are specific to a concrete BP-network. Two well-trained BP-networks can produce very different sensitivities. If we want to use the sensitivity coefficients as a universal feature ranking measure, we have to average the results over a higher number of trained BP-networks or to use an ensemble model [41].

Contrary to filters and wrappers, embedded methods are not a universal tool. However, they can benefit from the skills of the chosen computational model, while reducing the computational costs of the wrapper methods. Namely the

methods based on the BP-networks are able to capture also non-linear relationships among input features and the outputs. Both embedded and wrapper methods tend to have higher capacity than filter methods and are therefore more likely to overtrain [41]. Thus the filter methods may perform better for very small training sets. Embedded and wrapper methods will eventually outperform filter methods as the number of training patterns increases.

3.3 Methods for structure optimization

As already discussed, the topology of a BP-network is a very important optional parameter, that significantly affects both the performance and generalization abilities of the model. Topology of the network should correspond to the complexity of the analyzed data. If the BP-network is too small, it is not able to learn the task properly. On the contrary, if it is too large, it has a tendency to overtrain – it memorizes the training patterns but is not able to recognize patterns outside of the training set. Therefore, when training a larger network, we also need more training patterns to avoid overtraining.

There are several reasons for searching optimal (i.e., minimal) structure of BP-networks. The main goal is to improve and speed up prediction and to achieve better generalization. We appreciate the effectiveness of computation especially when working with large data sets. Other objectives are to decrease sensitivity to noisy data and to detect and manage the overtraining problem. Smaller networks also need less training patterns.

Nevertheless, searching for an optimal topology has more objectives than just an improved generalization and computational efficiency. It can also help create a clear and transparent structure of the network. This simplifies the following knowledge extraction. Moreover, some of the techniques for structure optimization are able to identify significant input parameters and relevant hidden neurons or weights. When we knew, how the inputs impact the outputs and which inputs are more significant than others, we could understand better the internal structure of a BP-network and we could also easier explain the underlying process. Extracting knowledge from a smaller network with a clear and simple structure is also much easier.

There are many different approaches to structure optimization, which can be divided into categories based on their principles and goals:

- Brute-force.
- Pruning algorithms.
- Network construction techniques.
- Probability optimization techniques.
- Regularization techniques.

Some of the methods just incrementally increase the number of neurons (e.g., network construction techniques), other techniques only decrease the network size (e.g., pruning algorithms), some adaptive methods enable both types of modifications (e.g., probability optimization techniques). In the following paragraphs, we will describe each approach in more detail.

3.3.1 Brute-force methods

Brute-force is the most common approach, how to find the optimal network size [94]. It is based on successive training of smaller networks, until the smallest topology is found, which still fits the data. This process is very time-consuming – many networks have to be trained. We also have to consider the remaining

initial parameters, because the behavior of the trained network is very sensitive to them.

There are also other problems connected with training of networks with topology, which is for a given task nearly optimal [43]. For such a network, it is difficult to converge and to learn the correct function. The smaller the network is, the more often it gets stuck in a local minimum during training. The training process is also usually much slower for networks with minimal topology than for larger ones.

3.3.2 Pruning algorithms

The principle of the pruning algorithms [94] is to train a larger network than necessary and then remove redundant parts of the final network until a reasonable topology is achieved. The usual training and pruning process can be described

Algorithm 3.3 General principle of pruning

1. *Training:* Train a BP-network with a reasonable fixed topology.
2. *Performance evaluation:* Compute the performance error on the validation data (the validation data should be disjoint from the training set).
3. *Pruning:* Repeat:
   (a) *Relevance evaluation:* Compute the relevance of hidden neurons, input neurons or weights using the chosen heuristic.
   (b) *Removal of insignificant elements:* Remove the least relevant element(s).
   (c) *Retraining:* Retrain the BP-network.
   (d) *Performance evaluation:* Compute the performance error on the validation data.

   Until a stop criterion for pruning is satisfied (e.g., the actual error on the validation data is lower than a given threshold, or it stops decreasing from one iteration of pruning to another.

in few steps, as shown in Algorithm 3.3.

The respective pruning algorithms differ in the way, how they solve the following important tasks:

1. Which elements of the network to remove?
2. When to stop pruning?
3. How to evaluate the significance or relevance of single network elements (i.e., weights or neurons)?
4. How to successively detect and remove the unimportant elements based on the chosen relevance measure?

Most of the pruning techniques target the pruning of edges (weights) or hidden neurons. A more difficult task is to remove also the irrelevant input neurons. Pruning of network inputs corresponds to the identification of as input features,

which are not needed for training (see Section 3.2). By their elimination we can obtain a network, which will generalize better [126].

Most of the pruning techniques establish a set of heuristic rules, that controls the pruning process. The heuristics must be created carefully – they should not work with a high number of optional parameters, the number of pruned elements must be reasonable (not too large or too small), and the pruning process should not be very costly.

Particular pruning methods use different strategies for measuring relevance. One of the most common relevance measures for weights is called weight saliency [62, 77]. Saliency is the sensitivity of the error function to the removal or change of a single weight. Actually, the removal of a weight corresponds to setting the weight equal zero. The sensitivity of weight w_i is formally defined as: $S(w_i) = E(w_i = 0) - E(w_i = w_i^f)$ where w_i^f is the final value of weigh w_i after training. Weight saliency is a very sophisticated and precise relevance measure, which is however highly inefficient [56, 77].

Several pruning methods approximate the weight saliency in order to improve computational efficiency (e.g., optimal brain damage (OBD) [62], optimal brain surgeon (OBS) [47] and others [77], [56]). In the case of OBD, the saliency is approximated by the second derivative of the error function with respect to the weight, where OBD assumes that the error function is quadratic and that the Hessian is diagonal. The OBS method is motivated by OBD, but differs in the way, in which the saliency is approximated. The OBS method iteratively computes the full Hessian, which leads to a more accurate approximation of the error function. The main weakness of the OBD and OBS techniques is their relatively low computational efficiency. Nevertheless, many modifications and approximations have been introduced – e.g., in [7], which try to deal with this problem.

The previous methods targeted pruning of weights. Hagiwara in [43] suggests three simple and effective strategies for detecting both redundant hidden neurons and weights (called Goodness factor, Consuming energy and Weights power). These measures are less sophisticated and precise than OBD or OBS, but require significantly less computational time.

The goodness factor is defined as:

$$G_i = \sum_p \sum_j (y_{pi} w_{ij})^2, \tag{3.45}$$

where p is an index over all training patterns, i is an index over all neurons in a fixed hidden layer, y_{pi} is their output, j is an index over all neurons in the next layer, and w_{ij} is the weight from the i-th to the j-th neuron. The goodness factor favors neurons with high absolute values of weights and activities for most patterns.

The so-called consuming energy is defined as it follows:

$$E_i = \sum_p \sum_j y_{pi} w_{ij} y_{pj}, \tag{3.46}$$

where y_{pj} is the output of the j-th neuron in the next layer. Consuming energy favors neurons, which excite frequently and at the same time as neurons in the next layer.

The so-called weight power subsumes only the weights and is defined as it follows:

$$W_i = \sum_j (w_{ij})^2. \tag{3.47}$$

Hagiwara in [43] has shown on the mirror symmetry problem, that all the three strategies massively reduce the network size and lead to better generalization, while the simplest strategy, Weight power, reaches the best results. However, the methods based on weight magnitude often remove also important parts of the network [47].

3.3.3 Sensitivity analysis

Generally, sensitivity analysis is a study of how to quantify the response of a computational model to parameter perturbations [123]. For BP-networks, there are two main approaches to sensitivity analysis [28, 123]:

1. with respect to the error function,
2. with respect to the BP-network's outputs.

The first approach – the error function sensitivity analysis – assesses the sensitivity of the applied error function to changes (or removals) of single weights and other parameters. Representatives of this approach were some of the previously discussed pruning methods (Saliency, OBD, OBS). Except weight pruning, the error function sensitivity can also be used to develop sophisticated training algorithms with improved convergence and stability [12, 57]. Other application is the study of the robustness and stability of the BP-networks – study of the conditions, under which the outputs of the BP-network change [4, 88].

The second approach – the output sensitivity analysis – evaluates the impact of small perturbations of the BP-network inputs and other parameters on its outputs. It has a wide range of applications [28, 123]: e.g., to study the generalization abilities of BP-networks [22], to measure the non-linearity of the training data [60], to detect and visualize decision boundaries [31], for selective learning [30, 51], to assess the significance of the input features [29], and for pruning of irrelevant input features and hidden neurons [28, 29, 34, 126].

The approaches to output sensitivity analysis differ in the way, how they compute the sensitivity coefficients. Tchaban et al. [108] suggested a simple, yet less precise sensitivity measure called weight product. For a neuron i and a neuron j from the immediately following layer that are connected by the weight w_{ij}, and for a training pattern p, the weight product S_{ij}^p is defined by:

$$S_{ij}^p = \frac{x_{pi} w_{ij}}{y_{pj}}, \tag{3.48}$$

where x_{pi} denotes the i-th input element of the neuron j and y_{pj} corresponds to its output value. Empirical studies [36, 115] have, however, shown that the approach of Tchaban et al. [108] is ineffective when examining the influence of the inputs on outputs.

A different definition of sensitivity in BP-networks has been suggested by Zurada et al. [126] and Fidalgo [34]. The sensitivity coefficients S_{ij} are computed

as the mean absolute value of the derivatives of the j-th output with respect to the i-th input over all of the P input patterns:

$$S_{ij} = \frac{1}{P} \sum_{p=1}^{P} \left| S_{ij}^{p} \right| = \frac{1}{P} \sum_{p=1}^{P} \left| \frac{\partial y_{pj}}{\partial x_{pi}} \right| . \tag{3.49}$$

This principle can be generalized to measure the sensitivity S_{ij}^{p} of the activity of a neuron j (in the output or in a hidden layer) to the activity of a neuron i (in one of the preceding layers).

Anyway, the coefficients S_{ij}^{p} can simply be computed recursively from the input layer to the output layer (or analogically in the opposite direction):

$$S_{ij}^{p} = \frac{\partial y_{pj}}{\partial y_{pi}} = f'(\xi_{pj})\, w_{ij} \tag{3.50}$$

for the sensitivity of the activity of a single neuron j to the activity of a neuron i in the preceding layer and

$$S_{ij}^{p} = \frac{\partial y_{pj}}{\partial y_{pi}} = \sum_{k} S_{kj}^{p} S_{ik}^{p} = \sum_{k} f'(\xi_{pj}) w_{kj} S_{ik}^{p} \tag{3.51}$$

for the sensitivity of the activity of the neuron $j \in L_{l_j}$ to the activity of the neuron $i \in L_{l_i}$, $l_j > l_i + 1$. k indexes the neurons in the hidden layer preceding j.

Such sensitivity measure S_{ij}^{p} can be used to evaluate the significance of input features, hidden neurons or input patterns with respect to the model outputs. However, we must take into consideration, that the sensitivity coefficients are computed separately for each training pattern. The significance of the input features for the outputs can be very different for various training patterns. An input feature (or a hidden neuron) can be assigned as superfluous only if the corresponding sensitivity coefficients are low for all the training patterns. Naturally, the sensitivity coefficients are specific to a concrete BP-network. Two well-trained BP-networks can produce very different sensitivities.

Several variants of sensitivity analysis use the computed sensitivity coefficients for pruning. In this respect, the key questions refer to the detection of redundant neurons and to the stop criteria for pruning. A. P. Engelbrecht, I. Cloete, J. M. Zurada and others [27, 29, 34, 126] use heuristics, which remove inputs and hidden neurons with the smallest sensitivity, Engelbrecht [28] suggests to remove neurons with minimum variance of sensitivity. Yeh and Cheng [122] use both the first- and second-order partial derivatives of the output variables with respect to the input variables.

Pruning based on the sensitivity coefficients may improve the generalization ability of the trained BP-networks [34]. Unfortunately, the sensitivity criteria alone are not capable of detecting all redundant neurons, because they assume that inputs and activities of hidden neurons are mutually independent and numerical. J. J. Montaño and A. Palmer [76] presents an extension called Numeric sensitivity analysis, which handles also quantitative or discrete input parameters.

The sensitivity coefficients can also be used for knowledge extraction from the BP-network model – we can, e.g., divide the training patterns into equivalence classes or clusters characterized by similar sensitivities [122].

Sensitivity analysis used for training

The concept of network sensitivities can be used also for learning. The Sensitivity-Based Linear Learning Method (SBLLM) [19] proposed for BP-networks with one hidden layer, calculates the weights by solving a system of linear equations. Assuming that the applied non-linear transfer functions are invertible, the method minimizes for each training pattern the difference between the actual and desired neuron potentials. For both the hidden and output neurons, the desired potential value can be determined as the value of the inverse transfer function of their desired activities.

During training, the error Q of the network is evaluated as the sum of the error Q^{HID} (determined as the difference between the desired and actual potential values over all hidden neurons) and Q^{OUT} (determined as the difference between the desired and actual potential values over all output neurons):

$$Q \;=\; Q^{HID} + Q^{OUT}\,, \tag{3.52}$$

$$Q^{HID} \;=\; \sum_p \sum_h \left(\sum_i w_{ih} x_{pi} - f_h^{-1}(y_{ph}) \right)^2 \,, \tag{3.53}$$

$$Q^{OUT} \;=\; \sum_p \sum_j \left(\sum_h w_{hj} y_{ph} - f_j^{-1}(y_{pj}) \right)^2 \,, \tag{3.54}$$

where p indexes the training patterns, i is the index of all the input neurons, h indexes all the hidden neurons and j indexes all the output neurons, f_h is the transfer function corresponding to neuron h.

This leads to a system of linear equations to be solved for both considered layers. Afterwards, the sensitivity terms for the sensitivity of the error Q^{HID} of the hidden neuron potentials to actual activities of the hidden neurons $\frac{\partial Q^{HID}}{\partial y_{ph}}$ and for the sensitivity of the error Q^{OUT} of the output neuron potentials to actual activities of the hidden neurons $\frac{\partial Q^{OUT}}{\partial y_{ph}}$ are computed and then also used to adjust the estimated "desired" activities for the hidden neurons:

$$\frac{\partial Q^{HID}}{\partial y_{ph}} \;=\; -\,\frac{2\left(\sum_i w_{ih} x_{pi} - f_h^{-1}(y_{ph})\right)}{f_h'(y_{ph})}\;;\quad \text{for all } p, h, \tag{3.55}$$

$$\frac{\partial Q^{OUT}}{\partial y_{ph}} \;=\; 2 \sum_j \left(\sum_h w_{hj} y_{ph} - f_j^{-1}(y_{pj}) \right) w_{hj}\;;\quad \text{for all } p, h. \tag{3.56}$$

These sensitivities allow then for an efficient and extremely fast iterative gradient-based update of the "desired" hidden neuron activities to be applied.

This method reaches a minimum error in a few epochs of training. This behavior is very convenient when dealing with huge data sets and large networks. However, when the training set is not representative enough, the few iterations employed by the method make it very difficult to avoid overtraining with techniques like early stopping.

A usual technique to avoid over-training is regularization that consists in adding a penalty term to the loss function. Therefore, a generalization of the SBLLM method [39] uses a regularization term based on the weight decay regularizer that is defined as the sum of squares of all the weights and thresholds in

the network. As a result, the weights of both layers are calculated independently by minimizing a new error function $\hat{Q}^{(l)}$ for each of the respective layers, l:

$$\hat{Q}^{(l)} = L^{(l)} + \alpha \sum_i \sum_j w_{ij}^2 \quad , \tag{3.57}$$

where

$$\begin{aligned} L^{(l)} &= \sum_p \sum_j \left(f_j'(y_{pj}) \varepsilon_{pj} \right)^2 \\ &= \sum_p \sum_j \left(f_j'(y_{pj}) \left(\sum_i w_{ij} x_{pi} - f_j^{-1}(y_{pj}) \right) \right)^2 \quad . \end{aligned} \tag{3.58}$$

In the above equations (3.57) and (3.58), α is the regularization parameter, the second term on the right-hand side of (3.57) is the regularization term, and i, and j are the indexes over the inputs and outputs of the considered layer l, y_{pj} is the desired output for the neuron j.

The term $L^{(l)}$ measures the training error also as the sum of squared errors before the non-linear transfer functions. However, as big differences of the potentials matter more around zero desired potentials, a scaling term to the sensitivity loss function has been introduced that multiplies it by the derivative of the transfer function applied to the desired layer output, $f_j'(y_{pj})$. This equalizes the errors calculated before and after the non-linearities.

The method of Zhong et al. [125] employs the output sensitivity of a binary BP-network to its parameter variation. The neuron with the highest sensitivity (to the output) will be selected for training, because in this way, the output can be altered the most.

Although the three above-sketched sensitivity-based techniques exhibit fast convergence, they are primarily aimed at training instead of feature selection and pruning. Due to the character of the networks found by solving a system of linear equations, the first two methods might also tend to overtrain [19, 125].

3.3.4 Network construction techniques

Network construction techniques start with a small network and incrementally add hidden neurons and weights until a reasonable topology is achieved [5, 6]. A similar approach is to split existing neurons if the performance of the network is not sufficient [66]. Network construction methods are usually not based on training from examples, as they require different training paradigms, such as genetic algorithms. Therefore, they are hardly comparable to the standard pruning or regularization techniques.

The best-known example of this approach is cascade correlation [33]. This method has several advantages: the training process is very quick, the network determines size and topology without outside interventions, and the algorithm is robust to changes of the training set during training. The key problems, crucial for the success of cascade correlation and other network construction techniques, are to decide, when to stop adding new neurons and when and in which place to add a new neuron or edge. A wrong treatment of these questions can lead to overtraining and strong slow-down of the construction process.

3.3.5 Probability optimization techniques

Probability optimization techniques dynamically alter the BP-network topology already during training. They are usually based on genetic algorithms or simulated annealing. Particular methods differ in the way, in which they code the solution.

A method introduced by Whitley et al. [120] trains a fully connected BP-network, then prunes weights using genetic operators and finally retrains the BP-network. In one iteration of the algorithm, more instances of the pruned and retrained network compete for survival and are awarded for fewer parameters and better generalization.

An advantage of these techniques consists in the fact, that they usually lead to good generalization, even if the training set is relatively small [44]. The main disadvantage is that the pruning process is very time-consuming.

3.3.6 Regularization techniques

Regularization techniques try to eliminate redundant weights already during training. A penalty term added to the error function usually enforces a decrease of absolute weight values during training. Weights smaller than a given threshold are regarded as useless and removed.

The well-known representative of this approach is called weight decay [119]. The penalty term added to the error function has the form

$$F \;=\; \beta \sum_i w_i^2, \tag{3.59}$$

where $0 < \beta \ll 1$ is a constant and i indexes all the weights and thresholds in the BP-network.

Each weight w_i is altered using:

$$w_i(t+1) \;=\; w_i(t) - \alpha \frac{\partial E_t}{\partial w_i} - \beta \frac{\partial F}{\partial w_i} \;=\; w_i(t) - \alpha \frac{\partial E_t}{\partial w_i} - \beta w_i(t), \tag{3.60}$$

where α, β are positive constants, E_t is the standard error function of the BP-algorithm corresponding to the training pattern that is presented to the BP-network in time t (see Eq. (1.8), (1.11), and (1.12)). The main idea of this process is, that unimportant weights have $\frac{\partial E_t}{\partial w_i} \simeq 0$ and therefore their values decrease exponentially:

$$w_i(t) \;\approx\; \beta^t w_i(0) \text{ for } 0 < \beta \ll 1. \tag{3.61}$$

A drawback of this method is that it favors weight vectors with many small elements to weight vectors with a few large elements, although the BP-network with the configuration of the second type may perform and generalize better [95].

Another regularization technique is weight elimination [117]. It uses the following penalty term:

$$F \;=\; \lambda \sum_i \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2}, \tag{3.62}$$

where λ and w_0 are problem-dependent constants, that have to be chosen carefully. The penalty term measures the complexity of the BP-network as a function of weight magnitudes related to the parameter w_0.

Other variants of the penalty terms – e.g. [36, 111] – yield varying results. Their main weakness consists in their relatively high computational costs and worse generalization [45].

3.3.7 Remarks

Besides structure optimization techniques, other methods can be used to improve the generalization, create simpler and smoother network function and thus enable simpler knowledge extraction – namely the methods of learning from hints and learning internal representation. These techniques will be described in the following Sections 3.4 and 3.5. Such methods can be easily combined with some of the techniques for structure optimization to achieve better results.

Analysis of the structure optimization methods

The above-discussed approaches to architecture optimization differ in many ways – they employ different principles, characteristics, and even goals. Some of the methods just try to shrink the network size and speed up the recall (e.g., regularization techniques). Other techniques are capable of improving generalization and robustness to noisy data (e.g., pruning algorithms and probability optimization techniques). More sophisticated strategies (e.g., sensitivity analysis) rather identify significant network elements and facilitate knowledge extraction.

Each of the described methods has its advantages but also drawbacks. No one can be denoted as the best in any situation and for all reasons. For example, regularization techniques can be easily and efficiently combined only with gradient-based training algorithms, such as back-propagation. On the other hand, network construction techniques usually require different training paradigms, such as genetic algorithms. Some of the pruning algorithms (OBS, weight magnitude, and sensitivity analysis) can be used with both types of learning algorithms.

A significant drawback of most standard methods consists in their low efficiency. Real-world applications therefore prefer simpler and more efficient methods, such as weight magnitude. Even though more sophisticated methods reach better results, precision is usually compensated by disproportional increase in computation time. The best results are thus often obtained by a combination of several applicable methods.

3.4 Methods for improved generalization

Although the standard BP-algorithm has the potential to generalize remarkably well, this ability strongly depends on many aspects (e.g., careful choice of model parameters and preprocessing of the training set). Additional techniques are thus often needed to assure sufficient generalization and to avoid overtraining [95]. Beside methods for structure optimization, that were described in Section 3.3, there are other techniques helpful in this respect (e.g., cross-validation and early stopping, learning with hints, training with jitter). In this section, we will discuss some of these methods in more detail.

3.4.1 Early stopping

A common heuristic approach to avoid overtraining is the so-called early stopping. It is based on the observation, that during the training the generalization error (see Section 1.3) typically tends to decrease firstly similarly to the performance error on the training data. However, after some time it reaches a minimum and begins to increase, while the error on the training set continues decreasing and the model overtrains.

Early stopping tries to identify the iteration of best generalization. It is a stop criterion based on the performance error on the validation data. The original training set T is divided into two disjoint parts: the new training set T_1 and the so called validation set V. The BP-network is trained on T_1, while V is used to assess, whether the overtraining has begun. In each iteration, we compute the performance error on V. We stop training as soon as the error on V is growing in $k \geq 1$ consecutive steps and return the model with the lowest validation error.

An advantage of early stopping is, that it often speeds up the training process, because it reduces the number of epochs. It also often helps to avoid overtraining [42]. A problem is, that for many tasks, the validation error doesn't follow the simple trend assumed by the method. It may not decrease monotonically and the local minima can simply confuse the method to stop too soon [10, 95]. The method may also be less effective for second-order training algorithms that are characterized by relatively high weight changes per iteration [10, 95].

3.4.2 Learning from Hints

Learning from hints was originally proposed by Abu-Mostafa [2, 3]. It is a powerful method for incorporating prior knowledge as a learning aid into the supervised training process.

A hint is any piece of information about the unknown function except the training set. It may take the form of naturally or even artificially generated input-output patterns or the form of a global constraint of the unknown function. Hints may have even local character or they may be related to the model architecture.

A hint may be helpful in the training process in two main ways: a) It may reduce the number of functions that are candidates to be the wanted network function. That may result in better generalization. b) It may reduce the number of steps necessary to find an appropriate approximation of the wanted function. In [2, 3], Abu-Mostafa has shown that restricting the space of candidate hypotheses for the wanted network function by learning from hints can reduce the VC-

dimension of the final network structures. Consequently, a hint may reduce the number of training patterns needed to learn the solution.

In the following paragraphs, we will describe two systematical ways, how to integrate hints into the training process: the extra examples hint method [1] and the extra output hint method [38]. An advantage of these two approaches is, that they present a general guideline for the treatment of almost any type of hints. Moreover, these methods are easy to implement. Unlike for most other hint methods, they don't require changes of the chosen training method or error criterion and thus they can be easily applied both to the standard BP-algorithm and its modifications like SCG and others. When solving a concrete task, the choice of one of these approaches depends on the character of the data and on the hints that are available.

Extra examples hint method The extra examples hint method was proposed by Abu-Mostafa in [1]. The principle of this approach is to represent hints in the form of input-output patterns. Such hints can become a part of the training set. Important questions are: a) How to transform hints into the training patterns? b) How many patterns to create for each hint? The training set should be well-balanced to enable the BP-network to learn simultaneously both the wanted network function and the hints.

We will illustrate the process of generation of the new training patterns on a simple example. Let $F : \Re^n \rightarrow [-1, 1]$ be the wanted network function and a hint be the knowledge that F is odd, i.e. $F(-\vec{x}) = -F(\vec{x})$. Then for each original training pattern $(\vec{x}_p, \vec{d}_p)$, $p = 1, ..., P$, we create and add to the training set a new one: $(-\vec{x}_p, -\vec{d}_p)$. A similar principle can be applied also to more complex hints, such as monotonicity or invariance [1].

Abu-Mostafa in [1] and others [61] propose various sophisticated strategies that prescribe, in which order and frequency to present the training patterns including hints to the BP-network during the training process. The goal of these strategies is that the BP-network learns as many hints as possible as well as possible.

An example of a simple training strategy is to present the training patterns cyclically in so called batches [1]. A batch is a subset of training patterns that correspond to a single hint. The number of patterns in a batch may change during the training process reflecting the actual importance of the particular hint.

The success of the extra examples hint method depends on the quality of the hints available and on the chosen training strategy. Abu-Mostafa in [1] has shown, that if the hints and the training strategy were chosen adequately, this method would improve and speed up the training process and reduce the number of training patterns required.

Extra output hint method The extra output hint method was originally proposed by Suddarth and others in [38], [106], [124]. This method consists in supplying the network with additional target outputs during training which express some knowledge about the problem. The hints have the form of a function with the same domain as the wanted network function. Similarly to the extra examples hint method, no modification of the training algorithm or error criterion is involved and the extra added outputs can be removed after training is finished.

Let $F : \Re^n \rightarrow [-1,1]^m$ be the unknown function and let $h : \Re^n \rightarrow [-1,1]^{m'}$ be the chosen hint function. $T = \{ (\vec{x}_p, \vec{d}_p) \mid \vec{x}_p = (x_{p1}, \ldots, x_{pn}), \vec{d}_k = (d_{p1}, \ldots, d_{pm}), 1 \leq k \leq P \}$ denotes the 'classical' training set for F. The form considered for the training set T' of the hint H will be: $T' = \{ (\vec{x}_p, \vec{d'}_p) \mid \vec{x}_p = (x_{p1}, \ldots, x_{pn}), \vec{d'}_k = (d'_{p1}, \ldots, d'_{pm'}), 1 \leq k \leq P \}$ When applying the hint H during training, the modified network will have $m + m'$ output neurons. The number of its input and hidden neurons will remain the same. The extended training set T'' will correspond to: $T'' = \{ (\vec{x}_p, \vec{d''}_p) \mid \vec{x}_p = (x_{p1}, \ldots, x_{pn}), \vec{d''}_p = (d_{p1}, \ldots, d_{pm}, d'_{p1}, \ldots, d'_{pm'}), 1 \leq p \leq P \}$.

The resulting network comprises two structures sharing their hidden neurons but learning to implement two distinct functions. In this way, the network is forced to develop an internal representation supporting an adequate approximation of both functions.

If the hint function is well-chosen, it may reduce the size of the weight space that a BP-network has to search for an appropriate set of weights. Then the training process will be faster and the chance to find a simpler model that generalizes better will be higher. On the other hand, a poor choice of a hint may even reduce the generalization ability [23, 38, 106]. That happens especially if the target and hint functions don't share a common sub-function for the BP-network to find. Similar problem occurs, if the hint function is too complex, so the model is not able to approximate both functions at the same time. In the second case, a possible solution is to train a bigger BP-network topology with more hidden neurons.

A simple heuristic, of how to find a good hint, is that it should be relevant to the problem and not too complex [38]. Examples of such hints are the knowledge about monotonic regions of the target function [38] or a suitable clustering of the training data [23]. Especially clustering of the training data is a hint that can be used in a wide range of problems, even if no more task-specific hints are available. Clustered data provide an intrinsic equivalence class structure expected to yield improved generalization.

3.4.3 Training with Jitter

The idea of training with jitter is to add a small random noise to the training input patterns during the training. In such a case, we never present to the BP-network exactly the same input pattern twice, so the model cannot simply memorize the training data and overtrain. This is helpful especially for larger BP-networks and if the number of training patterns is limited. The further goal of training with jitter is to make the wanted network function simpler and smoother and less sensitive to noise in the data.

Training with jitter succeeded in improving the generalization ability of the trained BP-networks in many practical applications [73, 93, 104]. A disadvantage of the method is that it requires small learning rates and more training epochs in order to average over the noise [93]. Another problem is the choice of an appropriate noise variance [49] that significantly affects the success of the method.

Training with Static Jitter Unfortunately, the training with jitter cannot be applied to the training algorithms, where the learning rates or the step sizes

depend on the change of the error from one iteration to the next (e.g., adaptive learning rate methods, conjugate gradients methods). It would make the training process very unstable.

A possible solution is a static variant of learning with jitter. Its idea is to create a larger training set by adding a sufficiently large amount of training patterns with an added random noise. A big problem of this method is the choice of an appropriate size of the training set with noisy training patterns. Reed in [95] showed on practical examples, that in many cases the sufficient training set with noisy patterns was even 100-times greater than the original one, while also the required number of iterations was about $100 - 1000$-times higher than when training without jitter.

While this method seems to be very parameter-dependent and inefficient, it is also not as powerful to avoid overtraining and to create a smoother network function as training with dynamic noise [95].

3.4.4 Summary of Section 3.4

In this section, we described some of the methods, that improve the generalization ability of the BP-networks, especially early stopping, learning from hints and training with jitter. While early stopping is a simple yet efficient heuristic to prevent overtraining, the other two methods, similarly to the regularization techniques, try to reduce the complexity of the model by reducing the number of functions that are candidates to be the wanted function.

While training with jitter naturally makes the training process much slower, learning from hints can even reduce the number of iterations needed. While training with jitter is restricted to just some of the training algorithms for BP-networks, especially the extra output hint method is not limited in this respect. However, success of the hint methods depends on the quality of the given hints, so choice of which method to use will always depend on the given task and other aspects.

3.5 Methods for creation of a transparent network structure

Standard BP-algorithm usually tends to create a non-transparent network structure. For such networks, it is not clear, what is the relation between the training data and the weights and activities of hidden neurons. Therefore it is extremely difficult to 'guess' the real meaning of every particular hidden or even input neuron for a proper network output. Such networks often use small differences of neuron outputs to distinguish between the presented patterns.

3.5.1 Learning condensed internal representation

A possible solution is the method of learning condensed internal representation (IR-algorithm) by Mrázová and Wang [86]. This regularization technique is designed for BP-networks with the sigmoidal transfer function.

The main idea of the IR-algorithm is to group the activities of the hidden neurons around three possible values – 0, $\frac{1}{2}$ and 1. Intuitively, the activity 1 corresponds to the active state of the neuron and to the value 'yes', 0 (the passive state) has the meaning 'no', and $\frac{1}{2}$ (the silent state) means 'don't know'. Such internal structure of the BP-network is equivalent to rules and enables an easy interpretation of the extracted knowledge, especially if the number of equivalence classes, given by the states of the hidden neurons, is not high.

The criterion for developing a transparent internal network structure is formulated as an additional term of the error function to be minimized during training – the representation error function.

Definition 8. For a given BP-network with h hidden neurons and an input pattern $\vec{x} \in \Re^n$, we call the vector $\vec{r} \in \Re^h$ of activities of all hidden neurons the internal representation of the given BP-network.
The internal representation $\vec{r} = (r_1, \ldots, r_h)$ of the given BP-network is called binar, if $r_i \in \{0, 1\}$ for $i = 1, \ldots, h$. It is called condensed, if $r_i \in \{0, \frac{1}{2}, 1\}$ for $i = 1, \ldots, h$.

Definition 9. For a given BP-network and a training set T, the hidden neuron i with the sigmoidal transfer function f_i forms:

1. the uniform representation, if $f_i(\xi_{pi}) = c \in \Re$ for all input patterns from T (indexed by p).

2. the representation identical to the representation of the hidden neuron j, if $f_i(\xi_{pi}) = f_j(\xi_{pj})$ for all input patterns from T (indexed by p).

3. the representation complementary to the representation of the hidden neuron j, if $f_i(\xi_{pi}) = 1 - f_j(\xi_{pj})$ for all input patterns from T (indexed by p).

The aim of the IR-algorithm is to develop a condensed internal representation of the BP-network or of its part during the training. Mrázová and Wang in [86] concentrated on the last hidden layer, i. e., the layer immediately connected to the output layer. In [80], Mrázová generalized the IR-algorithm also to other

hidden layers. Furthermore, the authors restrict the method to the model of BP-network with sigmoidal transfer function.

In the following paragraphs, we will discuss the form of the representation error function of the IR-method and show the derivation of the rules for weight adjustment. The representation error function should penalize hidden neurons with activities far from the values 1, 0 and $\frac{1}{2}$. It should be differentiable and non-negative, with local minima corresponding to hidden neuron activities equal to 1, 0 and $\frac{1}{2}$. This could be achieved by including the multiplicative terms $(1-y)$, y and $(y-\frac{1}{2})^2$ into the representation error function. These terms achieve their minima on the interval $[0,1]$ in the values – 1, 0 and $\frac{1}{2}$, respectively.

Figure 3.5 shows the graph of the term $y\,(1-y)\,(y-\frac{1}{2})^2$. Although this

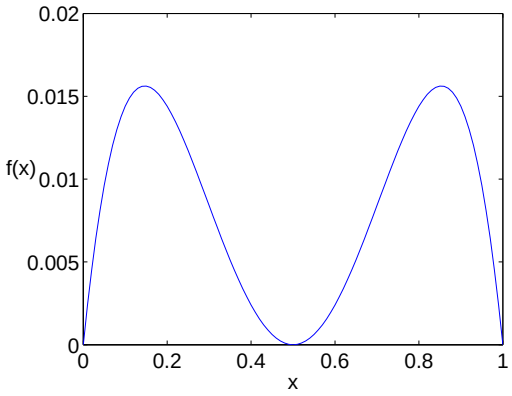


Figure 3.5: Graph of the function $x\,(1-x)\,(x-\frac{1}{2})^2$ on the interval $[0,1]$.

function meets all the given requirements, there is a problem: The local minima 0 and 1 are disadvantaged with regard to $\frac{1}{2}$. Maxima of this function are situated relatively close to 0 and 1, while there is a wide valley around the local minimum $\frac{1}{2}$. Such a penalty term would force hidden neurons to remain in silent states for most of the training patterns.

To move the two distant local maxima closer together, the term $y\,(1-y)\,(y-\frac{1}{2})^2$ can be replaced by the term $y^s\,(1-y)^s\,(y-\frac{1}{2})^2$ with $s>1$. Mrázová and Wang [86] recommend to set $s=4$. Such a penalty term would approach the local minima at 0 and 1 much smoother, as shown in Figure 3.6.

In sum, the wanted representation error function F is formulated as it follows:

$$F \;=\; \sum_{p} \sum_{j'} y_{pj'}^{s}\,(1-y_{pj'})^{s}\,(y_{pj'}-\frac{1}{2})^2 \;=\; \sum_{p} F_p \tag{3.63}$$

where p is an index over all training patterns and j' an index over all hidden neurons. y represents their activity and s is a parameter for tuning the shape of the representation error function. F_p is the representation error function corresponding to the p-th training pattern.

The overall error function of the IR-method has the form:

$$G \;=\; E + c_F\,F \tag{3.64}$$

where E represents the standard error function of the BP-algorithm (see Eq. 1.8) and F stands for the above defined representation error function. The coefficient

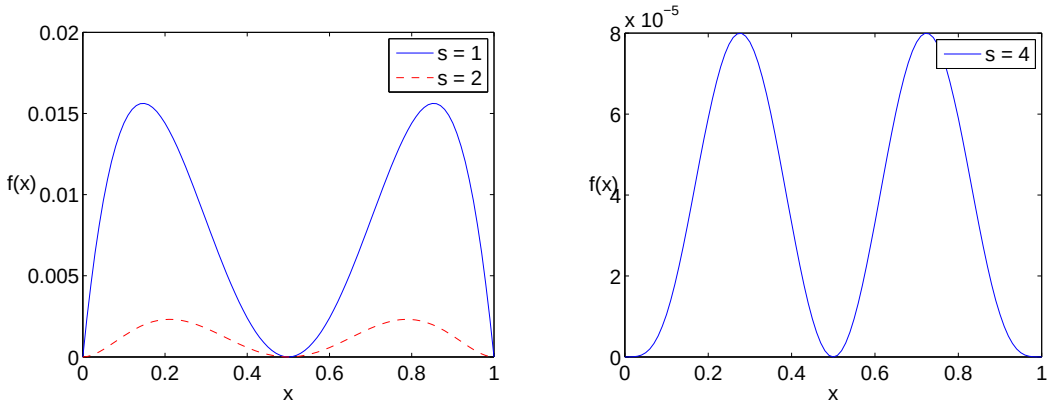


Figure 3.6: Graph of the function $x^s\,(1-x)^s\,(x-\frac{1}{2})^2$ on the interval $[0,1]$ for $s=1,2$ (on the left) and $s=4$ (on the right).

$c_F \geq 0$ reflects the trade-off between the influence of E and F in G. Its right choice can be crucial for the quality of the solution obtained. Small values of c_F will prevent the BP-network to form a condensed internal representation during the training. On the other hand, too large values of c_F can result into BP-networks with a perfectly formed condensed internal representation but incapable of approximating the desired function because of saturated hidden neuron outputs.

The training process of the IR-method is identical to the standard BP-algorithm, however with G as the error function. During the training process, both E and F are minimized simultaneously. The terms for adjusting the weights with respect to the error function E were stated in equations (1.17) and (1.18). To minimize also the representation error function F, the basic idea of gradient descent is applied as well. Let $(\vec{x}_p, \vec{d}_p)$ be the training pattern presented to the BP-network in time t. The change of the weight w_{ij} in time t corresponding to F will be denoted as $\Delta_F w_{ij}(t)$. It is proportional to the negative partial derivative of F_p with respect to this weight:

$$\Delta_F w_{ij}(t) \;\simeq\; -\frac{\partial F_p}{\partial w_{ij}}. \tag{3.65}$$

In the derivations below, neurons from the layer above the neuron j will be declared as those indexed by k. The actual output value and the potential of a neuron j will be denoted as y_{pj} and ξ_{pj}, respectively. Then, the terms for $\frac{\partial F_p}{\partial w_{ij}}$ can be derived in the following way:

- For neuron j from the output layer L_{l+2}:

$$\frac{\partial F_p}{\partial w_{ij}} \;=\; 0. \tag{3.66}$$

- For neuron j from the last hidden layer L_{l+1}:

$$\frac{\partial F_p}{\partial w_{ij}} \;=\; \frac{\partial F_p}{\partial w_{ij}} \;=\; \frac{\partial F_p}{\partial y_{pj}}\,\frac{\partial y_{pj}}{\partial \xi_{pj}}\,\frac{\partial \xi_{pj}}{\partial w_{ij}} \;=\; \frac{\partial F_p}{\partial y_{pj}}\,f'(\xi_{pj})\,y_{pi} \;=\; \varrho_{pj}\,y_{pi}, \tag{3.67}$$

  where $\varrho_{pj} \;=\; \frac{\partial F_p}{\partial \xi_{pj}} = \frac{\partial F_p}{\partial y_{pj}}\,\frac{\partial y_{pj}}{\partial \xi_{pj}} = \frac{\partial F_p}{\partial y_{pj}}\,f'(\xi_{pj})$ is an auxiliary term that can be denoted as the back-propagated representation error. For j from the last

hidden layer, $\frac{\partial F_p}{\partial y_{pj}}$ can be computed using the sum and product rules:

$$\begin{aligned}\frac{\partial F_p}{\partial y_{pj}} &= \frac{\partial}{\partial y_{pj}}\left(\sum_{j'} y_{pj'}^{s}\,(1-y_{pj'})^{s}\,(y_{pj'}-\frac{1}{2})^{2}\right) = \\ &= \frac{\partial\left[y_{pj}^{s}\,(1-y_{pj})^{s}\,(y_{pj}-\frac{1}{2})^{2}\right]}{\partial y_{pj}} = \\ &= {y_{pj}}^{s-1}\,(1-y_{pj})^{s-1}\,(y_{pj}-\frac{1}{2})\left[2\,(s+1)\,y_{pj}\,(1-y_{pj})-\frac{1}{2}\,s\right], \quad (3.68)\end{aligned}$$

where j' indexes all hidden neurons. Altogether, because $f'(\xi_{pj}) = \lambda\, y_{pj}\,(1-y_{pj})$:

$$\begin{aligned}\frac{\partial F_p}{\partial w_{ij}} &= \lambda {y_{pj}}^{s}(1-y_{pj})^{s}(y_{pj}-\frac{1}{2})\left[2(s+1)y_{pj}(1-y_{pj})-\frac{1}{2}s\right] y_{pi} = \\ &= \varrho_{pj} y_{pi}. \quad (3.69)\end{aligned}$$

- For neuron j from one of the other hidden layers $L_2, ..., L_l$:

  The equation (3.67) holds also for neurons j from hidden layers other than last. $\frac{\partial F_p}{\partial y_{pj}}$ will be computed indirectly using the sum and chain rules:

$$\begin{aligned}\frac{\partial F_p}{\partial y_{pj}} &= \frac{\partial\left[y_{pj}^{s}\,(1-y_{pj})^{s}\,(y_{pj}-\frac{1}{2})^{2}\right]}{\partial y_{pj}} + \sum_{q}\frac{\partial F_p}{\partial y_{pq}}\,\frac{\partial y_{pq}}{\partial \xi_{pq}}\,\frac{\partial \xi_{pq}}{\partial y_{pj}} = \\ &= {y_{pj}}^{s-1}\,(1-y_{pj})^{s-1}\,(y_{pj}-\frac{1}{2})\left[2\,(s+1)\,y_{pj}\,(1-y_{pj})-\frac{1}{2}\,s\right] + \\ &\quad + \sum_{q}\varrho_{pq}\,w_{jq}\,, \quad (3.70)\end{aligned}$$

  where q indexes neurons in the layer above the neuron j. Altogether:

$$\begin{aligned}\frac{\partial F_p}{\partial w_{ij}} &= \left\{{y_{pj}}^{s-1}\,(1-y_{pj})^{s-1}\,(y_{pj}-\frac{1}{2})\left[2\,(s+1)\,y_{pj}\,(1-y_{pj})-\frac{1}{2}\,s\right] + \right. \\ &\quad \left. + \sum_{q}\varrho_{pq}\,w_{jq}\right\}\lambda\, y_{pj}\,(1-y_{pj})\,y_{pi} = \\ &= \varrho_{pj}\,y_{pi}. \quad (3.71)\end{aligned}$$

The term (1.17) for weight adjustment will be altered in the following way:

$$w_{ij}(t+1) \;=\; w_{ij}(t) - \alpha\,\delta_{jp}\,y_{pi} - \alpha_r\,\varrho_{pj}\,y_{pi}, \quad (3.72)$$

where the term δ_{jp} was defined in Eq. (1.18) and

$$\varrho_{pj} = \begin{cases} 0, & j \in L_{l+2} \\ \lambda\, {y_{pj}}^{s}\,(1-y_{pj})^{s}\,(y_{pj}-\frac{1}{2})\,\left[2\,(s+1)\,y_{pj}\,(1-y_{pj})-\frac{1}{2}\,s\right], & j \in L_{l+1} \\ \lambda\, y_{pj}\,(1-y_{pj})\,\left\{\left[2\,(s+1)\,y_{pj}\,(1-y_{pj})-\frac{1}{2}\,s\right]\,{y_{pj}}^{s-1}\cdot\right. & \\ \quad \left.\cdot\,(1-y_{pj})^{s-1}\,(y_{pj}-\frac{1}{2})\,+\sum_{q}\varrho_r w_{jq}\right\}, & j \in L_k,\ 2 \le k \le l. \end{cases} \quad (3.73)$$

α and α_r from Equation (3.72) are constants representing the particular learning rates. λ is the parameter of the sigmoidal transfer function, s is the parameter for tuning the shape of the representation error function.

For the choice of the parameter $s = 4$, the formulated adaptation rules based on F lower down both the first- and second-order derivatives of the sigmoidal transfer function. The first-order derivative is equal $y' = y\,(1 - y)$. The second-order derivative is equal $y'' = y\,(1 - y)\,(1 - 2y)$. The term $y^4\,(1 - y)^4\,(y - \frac{1}{2})^2$ can be expressed in the form $\frac{1}{4}\,(y')^2\,(y'')^2$. The terms (3.72) and (3.73) for weight adjustment include both y' and y''. As a result, the hidden neurons are penalized during the training for absolutely high values of the first- and second-order derivatives of their transfer functions. In other words, the hidden neurons are punished for high curvature and great input-output sensitivity. In this way, the IR-method contributes to a smoother network function and better generalization ability of the trained model.

Pruning The IR-method also significantly supports subsequent pruning of redundant edges and neurons. Obviously, neurons providing identical internal representation can be more easily recognized and subsequently also removed with minor changes of the network – e.g. using the approach described by J. Sietsma and R. J. F. Dow [104]. The same holds for neurons yielding for all the patterns identical or complementary internal representation to another hidden neuron.

In the following paragraphs, we will describe the pruning method of Sietsma and Dow [104] in more detail:

- Let i be a neuron, that forms a uniform representation. So for a constant $c \in \Re$, its activities y_{pi} are equal to c for all training input patterns (indexed by p). To keep the same network outputs after the removal of the neuron i, it is sufficient to adjust the thresholds of all the neurons j in the following layer: $w_{0j}^{new} = w_{0j} + c\,w_{ij}$.

- Let i_1 and i_2 be two neurons from the same hidden layer that form identical representations to each other – their activities are identical for all training input patterns ($y_{pi_1} = y_{pi_2}$). We will combine these neurons into a single unit i. To keep the same network outputs after this change, we have to set the weights from i to all the neurons j in the following layer in the following way: $w_{ij}^{new} = w_{i_1 j} + w_{i_2 j}$.

- Let i_1 and i_2 be two neurons from the same hidden layer that have complementary activities for all training input patterns ($y_{pi_1} = 1 - y_{pi_2}$). Analogously to the previous case, we will combine these neurons into a single unit i. To keep the same network outputs after this change, we have to set the weights from i to all the neurons j in the following hidden layer and to adjust the thresholds of the neurons j in the following way: $w_{ij}^{new} = w_{i_1 j} - w_{i_2 j}$, $w_{0j}^{new} = w_{0j} + w_{i_2 j}$.

An advantage of this pruning technique is, that it needs no further retraining because the network outputs on the training data remain the same after pruning.

Remarks The main purpose of the IR-method is to create a transparent internal structure of the BP-network during the training process. The formed condensed internal representation is equivalent to rules and enables easier knowledge extraction from the model.

Another great advantage of the IR-method is, that it is expected to improve the generalization ability of the trained BP-networks, because it favors smoother network functions and facilitates further pruning of the trained model.

A disadvantage of the IR-method is its sensitivity to the parameters α and α_r that need to be tuned carefully. Poor choice of these parameters may result in a worsened prediction and generalization abilities of the trained model caused by overtraining or by saturation of the hidden neuron outputs. In this respect, combination of the IR-method with techniques that prevent overtraining (e.g. early stopping) and methods that improve the generalization ability (e.g. learning from hints) may be very helpful.

Further drawback of the IR-method are its relatively high time costs. They are caused mainly be the use of the standard BP-algorithm, which is relatively slow. However, the principle of the IR-method is general enough to be applied also to some of the faster training algorithms, which may largely solve the efficiency problem.

3.5.2 Learning unambiguous internal representation

An enhancement of the IR-method is the method of learning unambiguous internal representation (UIR-algorithm) [86]. This method forces the formed internal representation of the BP-network not only to be transparent, but also to differ as much as possible for substantially different output patterns.

Similarly to the IR-method, the method of learning unambiguous internal representation is a regularization technique that adds an additional term to the error function to be minimized during the training:

$$F^* = -\frac{1}{2}\sum_{p}\sum_{q\neq p}\sum_{j'}\sum_{o}(d_{po}-d_{qo})^2(y_{pj'}-y_{qj'})^2 \tag{3.74}$$

where p and q are indexes over all training patterns, j' an index over all neurons in the last hidden layer and o indexes all neurons in the output layer. y represents the activity of the hidden neurons and d represents the output patterns.

The UIR-algorithm is expected to improve generalization ability and decrease sensitivity of the trained BP-networks when compared to the IR-method. However, a great disadvantage of this method are its high time and space complexities, which are caused by the complexity of F^* and the corresponding rules for weight adaptation. After presenting each of the training pattern to the BP-network, it is necessary to compare it with all other training patterns. Moreover, it is needed to store the activities of all neurons in the last hidden layer for each training pattern.

4. Results achieved

4.1 Fast knowledge extraction

4.1.1 Introduction

The first goal – recalled In this section, we will design the first version of our framework for training of BP-networks, that will provide:

- **Speed:** A fast training algorithm, that doesn't have many tunable parameters and is robust to their choice.
- **Transparency:** Techniques that force the model to create a clear and transparent internal structure that will simplify knowledge extraction from the model.
- **Generalization:** Techniques that force the BP-network function to be smooth and generalize well.

In Chapter 2, we discussed the existing approaches to this goal and its subtasks. Now, we will briefly outline our approach and explain the reasons for our choice. A detailed description of our framework follows later in this section.

Speed: Among the fast training algorithms for BP-network training, we decided for the SCG-algorithm. Its advantages are low space complexity, robustness to noise and outliers in the data and a low number of tunable parameters. Its disadvantage is that it tends to create a complex and not transparent internal structure of the trained BP-network.

Transparency: To improve the transparency of the SCG-trained BP-networks, we decided for the IR-method. This regularization technique forces the BP-network to form a condensed internal representation during training. The condensed internal representation is equivalent to rules and enables an easier knowledge extraction from the model.

An advantage of this method is that it can be easily combined with many training algorithms, including SCG. Another great advantage of the IR-method is, that it is expected to improve the generalization ability of the trained BP-networks, because it favors smoother network functions and facilitates further pruning of the trained model [86].

A drawback of the original IR-method are its relatively high time costs that are caused mainly by the increased number of training epochs when compared to the standard BP-algorithm. However, the application of the IR-method to the SCG-algorithm may solve this problem.

Generalization: A disadvantage of the IR-method is its sensitivity to its parameters that need to be tuned carefully. A poor choice of parameters may result in worsened prediction and generalization abilities of the trained model. In this respect, a combination of the IR-method with techniques that prevent overtraining (e.g. early stopping) and methods that improve the generalization ability (e.g. learning from hints) may be very helpful. Learning from hints may also make the training process faster and improve the ability of the IR-method to create a

transparent structure of the BP-network. Knowledge extraction from the trained BP-network will be further supported by sensitivity analysis.

In the following paragraphs, we will describe our approach in detail.

4.1.2 Proposal of the SCGIR-method

In this section, we will describe the first version of our framework for training of the BP-networks called SCGIR [78, 82, 97]. It is based on the SCG-training algorithm (see Section 3.1.2) enhanced with the method of learning internal representation (described in Section 3.5) and with learning from hints (see Section 3.4.2).

IR-method for the 'bipolar' BP-network model

In this work, we decided for the so called 'bipolar' BP-network model, where all the hidden neurons have the hyperbolic tangent transfer function, while all the output neurons implement the linear transfer function. For the 'bipolar' BP-network model, the activities of the hidden neurons are from the interval $(-1, 1)$. This interval corresponds to the range of the hyperbolic tangent transfer function. Also the input patterns are usually normalized to the interval $[-1, 1]^n$, where n is the number of input features.

However, the IR-method, as proposed by the authors in [86] and [80], is designed for the 'binary' BP-network model, where all the hidden and output neurons implement the sigmoidal transfer function. Fortunately, the principle of the IR-method can be applied also to BP-networks with other continuous and differentiable transfer functions (including hyperbolic tangent and linear).

To alter the IR-algorithm to fit the 'bipolar' BP-network model [97], we have to define the representation error function and other concepts in a slightly different way than they were established in Section 3.5. We also have to alter the rules for weight adjustment to reflect the new representation error function and the derivatives of the applied transfer functions.

Definition 10. The internal representation $\vec{r} = (r_1, \ldots, r_h)$ of a BP-network with h hidden neurons is called bipolar, if $r_i \in \{-1, 1\}$ for $i = 1, \ldots, h$. It is called condensed, if $r_i \in \{-1, 0, 1\}$ for $i = 1, \ldots, h$.

Definition 11. For a given BP-network and a training set T, the hidden neuron i with the hyperbolic tangent transfer function f_i forms:

1. the uniform representation, if $f_i(\xi_{pi}) = c \in \Re$ for all input patterns from T (indexed by p).
2. the representation identical to the representation of the hidden neuron j, if $f_i(\xi_{pi}) = f_j(\xi_{pj})$ for all input patterns from T (indexed by p).
3. the representation complementary to the representation of the hidden neuron j, if $f_i(\xi_{pi}) = -f_j(\xi_{pj})$ for all input patterns from T (indexed by p).

The aim of the representation error function is to force the BP-network to develop a condensed internal representation during training. In the following paragraphs, we will propose its form for the model of 'bipolar' BP-networks.

Our goal is to group the activities of the hidden neurons around the values 1 ('yes'), -1 ('no') and 0 ('don't know'). Therefore, the error term corresponding to the activity y of a given hidden neuron will have the form $(1-y)^s\,(1+y)^s\,y^2$ with $s \geq 1$. Figure 4.1 shows the graph of this term for various values of the parameter s.

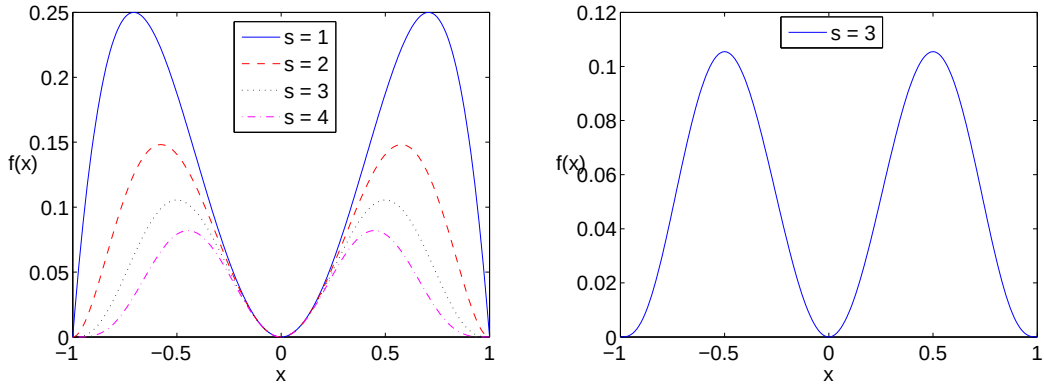


Figure 4.1: Graph of the function $(1-y)^s\,(1+y)^s\,y^2$ on the interval $[-1,1]$ for $s = 1, 2, 3, 4$ (on the left) and $s = 3$ (on the right).

The parameter s has to be set adequately to move the two distant local maxima farther from the values -1 and 1 towards 0. We decided to choose $s = 3$. For $s = 3$, the local maxima are closest to the values $-\frac{1}{2}$ and $\frac{1}{2}$, as shown in Figure 4.1.

The altered representation error function F [78] is given by the sum of the particular error terms over all hidden neurons (indexed by j') and over all training patterns (indexed by p):

$$F \;=\; F(\vec{w}) \;=\; \sum_p \sum_{j'} (1+y_{pj'})^s\,(1-y_{pj'})^s\,y_{pj'}^2 \;=\; \sum_p F_p(\vec{w})\,, \tag{4.1}$$

where $y_{pj'}$ is the activity of a neuron j' for the p-th training pattern, s is a parameter for tuning the shape of the representation error function. F_p is the representation error function corresponding to the p-th training pattern.

The overall error function G will have the form:

$$\begin{aligned} G = G(\vec{w}) &= E(\vec{w}) + c_F\,F(\vec{w}) = \\ &= \sum_p G_p(\vec{w}) \;=\; \sum_p \left[E_p(\vec{w}) + c_F\,F_p(\vec{w})\right] \;= \\ &= \frac{1}{2}\sum_p \sum_j (y_{pj}-d_{pj})^2 + c_F \sum_p \sum_{j'} (1+y_{pj'})^s (1-y_{pj'})^s y_{pj'}^2\,, \end{aligned} \tag{4.2}$$

where E represents the performance error function (defined by Equation 1.8) and F stands for the above defined representation error function. The coefficient $c_F \geq 0$ reflects the trade-off between the influence of E and F in G. c_F remains constant throughout the whole training process but might be variable, too. p goes over all training patterns, j and j' are indexes over all output neurons and hidden neurons, respectively. y denotes the activity of a neuron while d is its desired output value.

The derivation of the rules for weight adjustment with respect to the altered error function F (defined by (4.1)) [97] is analogical to the standard IR-method (see Section 3.5 and equations (3.66), ...,(3.71)), however with some differences. Let $(\vec{x}_p, \vec{d}_p)$ be the training pattern presented to the BP-network in time t and let $\Delta_F w_{ij}(t) \simeq -\frac{\partial F_p}{\partial w_{ij}}$ be the change of the weight w_{ij} in time t corresponding to F. In the following paragraph, we will describe the derivation of the term $\frac{\partial F_p}{\partial w_{ij}}$.

In the derivations below, neurons from the layer above the neuron j will be declared as those indexed by q. The activity and the potential of a neuron j will be denoted as y_{pj} and ξ_{pj}, respectively. The terms for $\frac{\partial F_p}{\partial w_{ij}}$ are derived in the following way:

- For neuron j from the output layer L_{l+2}:

$$\frac{\partial F_p}{\partial w_{ij}} \;=\; 0. \tag{4.3}$$

- For neuron j from the last hidden layer L_{l+1}:

$$\frac{\partial F_p}{\partial w_{ij}} \;=\; \frac{\partial F_p}{\partial y_{pj}}\,\frac{\partial y_{pj}}{\partial \xi_{pj}}\,\frac{\partial \xi_{pj}}{\partial w_{ij}} \;=\; \varrho_{pj}\, y_{pi}, \tag{4.4}$$

  where $\varrho_{pj} \;=\; \frac{\partial F_p}{\partial y_{pj}}\,\frac{\partial y_{pj}}{\partial \xi_{pj}}$ is the back-propagated representation error. It is computed differently to the standard IR-method. The first derivative is given by:

$$\frac{\partial F_p}{\partial y_{pj}} = \frac{\partial}{\partial y_{pj}}\left(\sum_{j'}(1+y_{pj'})^s(1-y_{pj'})^s y_{pj'}^2\right)\,, \tag{4.5}$$

  where j' indexes all hidden neurons. Because

$$\frac{\partial\left[(1+y_{pj'})^s(1-y_{pj'})^s y_{pj'}^2\right]}{\partial y_{pj}} = 0$$
$$\text{for all } j'\,,\ (j' \in L_{l+1}\,, j' \neq j)\ \text{ or }\ (j' \in L_k\,, 2 \leq k \leq l)\,, \tag{4.6}$$

  we can use the product rule and compute:

$$\begin{aligned}\frac{\partial F_p}{\partial y_{pj}} &= \frac{\partial\left[(1+y_{pj})^s(1-y_{pj})^s y_{pj}^2\right]}{\partial y_{pj}} \;= \\ &= s(1+y_{pj})^{s-1}(1-y_{pj})^s y_{pj}^2 \;+\; (1+y_{pj})^s(-s)(1-y_{pj})^{s-1}y_{pj}^2 \;+ \\ &\quad + (1+y_{pj})^s(1-y_{pj})^s 2y_{pj} \;= \\ &= 2\left[1-(s+1)\,y_{pj}^2\right]\,(1+y_{pj})^{s-1}\,(1-y_{pj})^{s-1}\,y_{pj}\,.\end{aligned} \tag{4.7}$$

  The derivative of the hyperbolic tangent transfer function is given by:

$$\frac{\partial y_{pj}}{\partial \xi_{pj}} = f'(\xi_{pj}) = (1+y_{pj})\,(1-y_{pj}) = (1-y_{pj}^2)\,. \tag{4.8}$$

  Altogether:

$$\frac{\partial F_p}{\partial w_{ij}} = 2\left[1-(s+1)\,y_{pj}^2\right]\,(1-y_{pj}^2)^s\,y_{pj}\,y_{pi} \;=\; \varrho_{pj} y_{pi}. \tag{4.9}$$

- For neuron j from a hidden layer $L_k\,, 2 \leq k \leq l$:

  The equations (4.4) and (4.5) hold also for neurons j from hidden layers other than last. Moreover,

$$\frac{\partial\left[(1+y_{pj'})^s(1-y_{pj'})^s y_{pj'}^2\right]}{\partial y_{pj}} = 0 \qquad \text{for all } j'\,,\ (j' \in L_k\,, j' \neq j) \ \text{ or } \ (j' \in L_{k'}\,, 2 \leq k' \leq k)\,. \tag{4.10}$$

  $\frac{\partial F_p}{\partial y_{pj}}$ will be computed indirectly using the sum and chain rules:

$$\begin{aligned}\frac{\partial F_p}{\partial y_{pj}} &= \frac{\partial\left[(1+y_{pj})^s(1-y_{pj})^s y_{pj}^2\right]}{\partial y_{pj}} + \sum_q \frac{\partial F_p}{\partial y_{pq}}\,\frac{\partial y_{pq}}{\partial \xi_{pq}}\,\frac{\partial \xi_{pq}}{\partial y_{pj}} = \\ &= 2\left[1-(s+1)\,y_{pj}^2\right]\,(1-y_{pj}^2)^{s-1}\,y_{pj} + \sum_q \varrho_{pq}\,w_{jq}\,,\end{aligned} \tag{4.11}$$

  where q indexes the neurons in the layer above the neuron j. Altogether:

$$\begin{aligned}\frac{\partial F_p}{\partial w_{ij}} &= \left[2\left[1-(s+1)\,y_{pj}^2\right]\,(1-y_{pj}^2)^{s-1}\,y_{pj} + \sum_q \varrho_{pq}\,w_{jq}\right]\,(1-y_{pj}^2)\,y_{pi} = \\ &= \varrho_{pj}\,y_{pi}.\end{aligned} \tag{4.12}$$

If we combine the altered IR-method with the standard BP-algorithm (we will call this method GDIR), the term for weight adjustment will be analogical to the IR-method (see Equation (3.72)):

$$w_{ij}(t+1) \ \simeq \ w_{ij}(t) - \frac{\partial G}{\partial w_{ij}}(t) \ = \ w_{ij}(t) - \alpha\,\delta_{pj}\,y_{pi} - \alpha_r\,\varrho_{pj}\,y_{pi}, \tag{4.13}$$

where the term δ_{pj} was defined in Eq. (1.18) and

$$\varrho_{pj} = \begin{cases} 0\,, & j \in L_{l+2} \\ 2\left[1-(s+1)\,y_{pj}^2\right]\,(1-y_{pj}^2)^s\,y_{pj}\,, & j \in L_{l+1} \\ 2\left[1-(s+1)\,y_{pj}^2\right]\,(1-y_{pj}^2)^s\,y_{pj} + & \\ \qquad + (1-y_{pj}^2)\,\sum_q \varrho_{pq} w_{jq}\,, & j \in Lk,\ 2 \leq k \leq l. \end{cases} \tag{4.14}$$

In the above equations, α and α_r are constants representing the particular learning rates, s is the parameter for tuning the shape of the representation error function, q indexes neurons in the layer above the neuron j.

The SCGIR-method

Our new SCGIR-method [78, 97] combines the altered IR-method with the SCG training algorithm. The training process of the SCGIR-method is identical to the standard SCG-algorithm, however with G (defined by (4.2)) as the error function. Algorithm 3.2 on page 44 will be modified in the following way: The

terms $\vec{r}_k = -E'(\vec{w}_k)$ will be replaced by the terms $\vec{r}_k = -G'(\vec{w}_k)$. For details, see Algorithms 4.1 and 4.2 on pages 79 and 80.

The derivative of the error function $G'(\vec{w}) = (..., \frac{\partial G}{\partial w_{ij}}, ...)$, can be computed as $G'(\vec{w}) = E'(\vec{w}) + c_F\, F'(\vec{w})$, where $\frac{\partial G}{\partial w_{ij}} = \frac{\partial E}{\partial w_{ij}} + c_F \frac{\partial F}{\partial w_{ij}}$.

To simplify the description, we will set $G'(\vec{w}) = \sum_p G'_p(\vec{w})$, where $G'_p(\vec{w}) = (..., \frac{\partial G_p}{\partial w_{ij}}, ...)$ is the derivative of the error function corresponding to the p-th training pattern, with the elements $\frac{\partial G_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial w_{ij}} + c_F \frac{\partial F_p}{\partial w_{ij}}$. They are computed in the following way:

$$\frac{\partial G_p}{\partial w_{ij}} = \begin{cases} (y_{pj} - d_{pj})\, y_{pi}\,, & j \in L_{l+2} \\ \Big\{ \sum_q \delta_{pq}\, w_{jq} \;+\; 2\, c_F\, [1 - (s+1)\, y_{pj}^2] \cdot \\ \qquad \cdot (1 - y_{pj}^2)^{s-1}\, y_{pj} \Big\} (1 - y_{pj}^2)\, y_{pi}\,, & j \in L_{l+1} \\ \Big\{ \sum_q (\delta_{pq} + \; c_F\, \varrho_{pq})\, w_{jq} \;+\; 2\, c_F\, [1 - (s+1)\, y_{pj}^2] \cdot \\ \qquad \cdot (1 - y_{pj}^2)^{s-1}\, y_{pj} \Big\} (1 - y_{pj}^2)\, y_{pi}\,, & j \in L_k,\ 2 \le k \le l. \end{cases} \tag{4.15}$$

$\vec{w}$ is the actual configuration of the BP-network, y denotes the activity of a neuron while d is its desired output value. q indexes neurons in the layer above the neuron j, s is the parameter for tuning the shape of the representation error function F. c_F is a constant representing the influence of F in G. δ_{pq} and ϱ_{pq} can be determined according to (1.18) and (4.14), resp.

Algorithm 4.1 Function $SCGIR()$ (Scaled conjugate gradients algorithm with learning internal representation):

---

1. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of size W.
   A training set $T = \{\, (\vec{x}_p, \vec{d}_p\,) \mid \quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$.
2. *Train:*
   Train the BP-network M on the training set T using the SCG-training algorithm (described by Algorithm 4.2 on page 80) and the overall error function $G(\vec{w}) = \sum_p G_p(\vec{w})$ defined by (4.2) with $\frac{\partial G_p}{\partial w_{ij}}$ defined by (4.15).
   $M' = SCG(M, T, G)$.
3. *Return:* M'.

Similarly to the standard IR-method, the choice of the parameter s for tuning the shape of the representation error function F is an important issue. We decided to set $s = 3$. For this choice, the formulated adaptation rules based on F minimize both the first- and second-order derivatives of the hyperbolic tangent transfer function. The first-order derivative is equal $y' = (1+y)\,(1-y)$. The second-order derivative is equal $y'' = -2\,y$. The term $(1+y)^3\,(1-y)^3\,y^2$ can be expressed in the form $\frac{1}{4}\,(y')^3\,(y'')^2$. Therefore, similarly to the standard IR-method (and $s = 4$), the hidden neurons are penalized during training for absolutely high values of the first- and second-order derivatives of their transfer functions. In other words, the hidden neurons are punished for high curvature and input-output sensitivity.

Algorithm 4.2 Function $SCG()$ (General schema of the SCG-like training algorithms)

0. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of size W.
   A training set $T = \{ (\vec{x}_p, \vec{d}_p) \mid \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$,
   An error function $Err(\vec{w})$ with the derivative $Err'(\vec{w})$.
1. *Initialization:*
   Set the discrete time variable $t = 1$ and set $success = true$.
   Initialize the weight vector $\vec{w}_1$ with small random values.
   Set scalars such that $0 < \sigma \leq 10^{-4}$, $0 < \lambda_1 \leq 10^{-8}$ and $\overline{\lambda}_1 = 0$.
   Calculate the direction based on the error function $\vec{r}_1 = -Err'(\vec{w}_1)$.
   Set the search direction $\vec{g}_1 = \vec{r}_1$.
2. If $success = true$, then calculate the second-order information:
   $$\sigma_t = \frac{\sigma}{\| \vec{g}_t \|} \quad ; \quad \vec{s}_t = \frac{Err'(\vec{w}_t + \sigma_t \vec{g}_t) - Err'(\vec{w}_t)}{\sigma_t} \quad ; \quad \delta_t = \vec{g}_t^{\,T} \vec{s}_t \quad .$$
3. Scale $\vec{s}_t$, δ_t:
   $$\vec{s}_t = \vec{s}_t + (\lambda_t - \overline{\lambda_t})\, \vec{g}_t \quad ; \quad \delta_t = \delta_t + (\lambda_t - \overline{\lambda_t}) \parallel \vec{g}_t \parallel^2 \quad .$$
4. If $\delta_t \leq 0$, then make the Hessian matrix positive definite by setting:
   $$\vec{s}_t = \vec{s}_t + \left(\lambda_t - 2\frac{\delta_t}{\| \vec{g}_t \|^2}\right) \vec{g}_t \quad ; \quad \overline{\lambda}_t = 2\left(\lambda_t - \frac{\delta_t}{\| \vec{g}_t \|^2}\right) \quad ;$$
   $$\delta_t = -\delta_t + \lambda_t \parallel \vec{g}_t \parallel^2 \quad ; \quad \lambda_t = \overline{\lambda_t} \quad .$$
5. Calculate step size α_t : $\quad \mu_t = \vec{g}_t^{\,T} \vec{r}_t \quad ; \quad \alpha_t = \frac{\mu_t}{\delta_t} \quad .$
6. Calculate the comparison parameter Δ_t :
   $$\Delta_t = \frac{2\,\delta_t\,[G(\vec{w}_t) - G(\vec{w}_t + \alpha_t \vec{g}_t)]}{\mu_t^2} \quad .$$
7. If $\Delta_t \geq 0$, then a successful reduction in the value of the error function G can be made:
   $$\vec{w}_{t+1} = \vec{w}_t + \alpha_t \vec{g}_t \quad ; \quad \vec{r}_{t+1} = -Err'(\vec{w}_{t+1}) \quad ;$$
   $$\overline{\lambda}_t = 0 \quad ; \quad success = true \quad .$$
   7.a If $t \bmod W = 0$, then restart the algorithm by setting: $\vec{g}_{t+1} = \vec{r}_{t+1}$,
   else create a new conjugate direction:
   $$\beta_{t+1} = \frac{\vec{r}_{t+1}^{\,T}(\vec{r}_{t+1} - \vec{r}_t)}{\| \vec{r}_t \|^2} \quad \text{and} \quad \vec{g}_{t+1} = \vec{r}_{t+1} + \beta_{t+1} \vec{g}_t \quad .$$
   7.b If $\Delta_t > 0.75$ then reduce the scale parameter: $\lambda_t = \frac{1}{4}\lambda_t$,
   else reduction in error is not possible:
   $$\overline{\lambda}_t = \lambda_t \quad ; \quad success = false \quad .$$
8. If $\Delta_t < 0.25$ then increase the scale parameter: $\lambda_t = \lambda_t + \frac{\delta_t (1 - \Delta_t)}{\| \vec{g}_t \|^2}$
9. If the stop condition is not satisfied (e.g., the steepest descent direction $\vec{r}_t \neq 0$), set $t = t + 1$ and go to step 2),
   else terminate and return the BP-network with the weight vector $\vec{w}_t$ as the final one.

In this way, also the SCGIR-method contributes to a smoother network function and better generalization ability of the trained model [16].

Further supplements of the SCGIR-method

A problem of the IR-method is its sensitivity to the choice of parameter c_F. Too small values of c_F may prevent the BP-network from forming a condensed internal representation during training. On the other hand, larger values of c_F may lead to worse generalization and approximation because of saturated hidden neuron outputs.

We assume that also the SCGIR-method will suffer from the c_F-sensitivity problem. However we expect that it will be less serious than in the case of the IR-method due to the robustness of the SCG-training algorithm. On the other hand, it may be more difficult to achieve a perfect transparent internal representation using the fast and robust SCG-algorithm than using the standard BP-algorithm – due to the reduced number of training epochs and greater step sizes.

Nevertheless, we decide to combine the proposed technique (Algorithm 4.1 on page 79) with further enhancements, that will support improved generalization and transparency:

Learning from hints To improve the generalization ability and stability of the trained networks, we decided to implement learning from hints, namely the extra output hint method [106]. As hints we use c-means clustering of the training data [23]. It is a general approach independent on the choice of the training algorithm and error function. Clustered data provide an intrinsic equivalence class structure expected to yield improved generalization.

We hope that learning from hints will help the SCGIR-method to create a condensed internal representation of the trained network and to identify redundant hidden and input neurons. Learning from hints may also make the training process faster. A further advantage of the extra output hint method is that no modification of the network algorithm or error criterion is involved.

Pruning based on the internal representation The IR-method together with learning from hints may help to simpler identification of redundant hidden neurons identified by uniform, identical or complementary representations. Such hidden neurons can be simply pruned using the approach described by J. Sietsma and R. J. F. Dow [104]. This pruning technique was described in detail in Subsection 3.5.1 on page 71. Its enhancement to the 'bipolar' BP-network model will be introduced in the following Section 4.2.2. This pruning technique needs no retraining of the once trained and pruned BP-network, because the network outputs on the training data remain the same after pruning.

Knowledge extraction based on the sensitivity analysis Knowledge extraction from the trained and pruned BP-network can be further supported by the sensitivity analysis. We decided to use it especially to detect and visualize decision boundaries [31] and to assess the significance of the input features [29].

4.1.3 Summary of Section 4.1

In this section, we proposed the first version of our framework – the SCGIR-method. It is intended to be fast and relatively insensitive to the choice of tunable parameters, to provide improved generalization, to create a transparent network structure and allow an easy interpretation of the extracted knowledge. The qualities and drawbacks of this method will be assessed in the experimental part of this work in Chapter 5. For more experimental results, see also [78, 82, 97].

4.2 Topology simplification

4.2.1 Introduction

The second goal – recalled In this section, we will enhance our framework for training of the BP-networks to fulfill also the second of our goals:

- **Simplification of the topology:** Creation of a simple yet adequate model topology.
- **Measuring relevance:** Automatic detection of important and irrelevant parts of the BP-network.
- **Feature selection:** Sophisticated selection of relevant input features.

Possible approaches to this task and its subtasks were discussed in Chapter 2. In this section, we will derive our approach to these tasks and explain the main reasons for our choice.

Among the techniques for structure optimization, we decided for pruning based on the formed internal representation and on the output sensitivity analysis [83, 85]. The main reasons for our choice are the following:

- Both methods are very efficient, generally usable and compatible with the techniques already included in our framework. They are intended to help the SCGIR-method to create a simple and transparent structure of the BP-network and thus simplify knowledge extraction from the model.
- Both methods should also contribute to smoother network functions and improved generalization ability of the trained model.
- Namely the sensitivity analysis also satisfies all of our requirements on structure optimization, feature selection and relevance measurement at once.

Moreover, the pruning method based on internal representation and the IR-regularization technique are supposed to mutually facilitate each other: not only that the creation of the condensed internal representation should be improved by pruning, but also that the IR-method should simplify pruning based on internal representation.

Similarly, to accelerate and simplify pruning based on sensitivity analysis, a well-designed regularization technique might be useful. Unfortunately, existing regularization methods are mostly focused on different tasks (e.g., weight minimization). For this reason, we decided to design and integrate into our framework a new analytical sensitivity-based (SC) regularization technique [83, 84, 85].

In the following subsections, we will describe our training and pruning approach in detail. Later in this section, we will introduce the new SC-regularization technique.

4.2.2 Pruning based on internal representation

The SCGIR-method together with learning from hints may supply to simpler identification of redundant hidden neurons identified by uniform, identical or complementary representations. Such hidden neurons can be simply pruned using the approach described by J. Sietsma and R. J. F. Dow [104]. An advantage of this pruning technique is high efficiency, because the network outputs on the training

data remain the same after pruning and no further retraining of the pruned BP-model is needed.

This pruning technique was originally proposed for the standard 'binary' BP-network model [104]. A detailed description of the pruning mechanism is stated in Subsection 3.5.1 on page 71. To alter the technique to fit the 'bipolar' BP-network model, we have to change slightly the pruning rules for the case of two hidden neurons with complementary representations to each other:

- Let i_1 and i_2 be two neurons from the same hidden layer that have complementary activities for all training input patterns (indexed by p): $y_{pi_1} = -\,y_{pi_2}$. These neurons will be combined into a single unit i. To keep the same network outputs after this change, we have to set the weights from i to all the neurons j in the following hidden layer in the following way: $w_{ij}^{new} = w_{i_1 j} - w_{i_2 j}$.

Pruning rules for the cases of two hidden neurons with identical representations to each other and for neurons with uniform representations are the same like for the 'binary' BP-network model. The pruning strategy based on internal representation is summarized in Step 2. of Algorithm 4.4 on page 90.

4.2.3 Pruning based on sensitivity analysis

The general principle of the so-called relevance-based pruning algorithms [94] is to train a larger network than necessary and then remove redundant parts of the final network until a reasonable topology and model performance is achieved. When designing a pruning technique, we can follow the following questions:

1. Which parts of the BP-network to prune and how to evaluate their relevance?
2. How to identify the redundant elements to be pruned?
3. How to control the pruning process (e.g., when to stop pruning)?

In the following paragraphs, we will present a detailed description of our sensitivity-based pruning algorithm [83, 85] through successive answering of these questions.

1. Which parts of the BP-network to prune and how to evaluate their relevance?

We decided to prune both the hidden neurons and input neurons that correspond to input features. This way we can manage feature selection as an integral part of the training-and-pruning process. As a relevance measure, we use the exact sensitivity coefficients S_{ij}^{p} suggested by Zurada et al. [126] and Fidalgo [34] (defined by Equations (3.50) and (3.51)). See Subsection 3.3.3 for a detailed description of this approach and its alternatives.

We preferred this sophisticated sensitivity-based relevance measure over the other ones (e.g., the so-called weight product [108] defined by Equation (3.48), or the so-called consuming energy [43] defined by Equation (3.46)) due to its exactness and accuracy. The exact sensitivity coefficients may be more likely to correctly identify the important and redundant parts of the BP-network than

their less precise yet computationally more efficient alternatives. Some of the approximative sensitivity measures (e.g., weight product [108]) may prune more neurons, however the prediction and generalization abilities of the pruned model may be worse.

In our case, we use the coefficient S_{iv}^p to measure the sensitivity of the activity of a neuron v in the output layer to the activity of a neuron i from one of the preceding layers. Because for the 'bipolar' BP-network model, $f'(\xi_{pv}) = 1$ for all the output neurons (indexed by v) and $f'(\xi_{ph}) = (\,1 - y_{ph}^2\,)$ for all the hidden neurons (indexed by h), S_{iv}^p can be determined as:

$$S_{iv}^p = \frac{\partial y_{pv}}{\partial y_{pi}} = f'(\xi_{pv})\, w_{iv} = \, w_{iv} \tag{4.16}$$

for the sensitivity of the activity of the output neuron $v \in L_{l+2}$ to the activity of a neuron i in the last hidden layer L_{l+1} and

$$S_{iv}^p = \frac{\partial y_{pv}}{\partial y_{pi}} = \sum_g S_{gv}^p S_{ig}^p = \sum_g S_{gv}^p f'(\xi_{pg}) w_{ig} = \sum_g S_{gv}^p (1 - y_{pg}^2\,) w_{ig} \tag{4.17}$$

for the sensitivity of the activity of the output neuron $v \in L_{l+2}$ to the activity of the neuron $i \in L_k$, $1 \leq k \leq l$. g indexes the neurons in the layer L_{k+1}. For the input neurons i, their activity $y_{pi} = x_{pi}$, where x_{pi} is the i-th input feature. The coefficients S_{ij}^p can be computed recursively (in the direction from the last hidden layer to the input layer).

A big advantage of the sensitivity relevance measure S_{iv}^p is, that it can be used in addition to pruning also for other purposes that simplify knowledge extraction from the BP-network model:

1. to assess the significance of input features [29],
2. to assess the significance of training patterns [51],
3. to detect and visualize decision boundaries [31].

The significance of an input feature u for the network outputs can be computed e.g., by:

$$S_u = \underset{\{p,v\}}{\text{mean}} \, |S_{uv}^p| \,, \tag{4.18}$$

where p indexes the training patterns, v indexes the network outputs and S_{uv}^p is the sensitivity of the output neuron v to the activity of the input neuron u for the input pattern p. The coefficients S_u can be used as a relevance measure to rank the input features [29].

Similarly, the significance of a training input pattern p for the network outputs can be computed by:

$$S^p = \underset{\{u,v\}}{\text{mean}} \, |S_{uv}^p| \,, \tag{4.19}$$

where u indexes the input features and v indexes the network outputs. The coefficients S^p can be used as a relevance measure to rank the training patterns [51], to detect the decision boundaries [31] and for selective learning [30]. The idea of these techniques is that the training patterns closest to the decision boundaries are the most informative and are characterized by high values of S^p [31].

A problem of the relevance measures S_u and S^p is, that they are specific to a concrete BP-network. To achieve more general results, we would have to normalize the values of S_u and S^p and average them over a higher number of trained BP-networks.

2. How to identify the redundant elements to be pruned?

Another question is, how to use the coefficients S_{iv}^p to identify superfluous neurons to be pruned. A problem of this relevance measure arises from the fact that the sensitivity coefficients are computed separately for each training pattern and for each network output. The significance of the input features and hidden neurons for the outputs can be very different for various training patterns and for various outputs. The existing pruning strategies [27, 28, 34, 122, 126] differ in the way, how they get over this problem.

A possible solution is to average the sensitivity coefficients over the training patterns in some way. Zurada et al. [126] concentrate only on input neurons. They suggest to compute the so called mean squared average sensitivity coefficients S_{uv}^* as the square root of the squared sensitivity coefficients averaged over the input patterns (indexed by p):

$$S_{uv}^* = \sqrt{\underset{p}{\text{mean}} \left(S_{uv}^p\right)^2} = \sqrt{\frac{1}{P} \sum_{p=1}^{P} \left(S_{uv}^p\right)^2} = \sqrt{\frac{1}{P} \sum_{p=1}^{P} \left(\frac{\partial y_{pv}}{\partial y_{pu}}\right)^2}. \tag{4.20}$$

The significance of an input neuron u over all outputs (indexed by v) is computed by:

$$S_u^* = \max_v S_{uv}^* = \max_v \sqrt{\frac{1}{P} \sum_{p=1}^{P} \left(\frac{\partial y_{pv}}{\partial y_{pu}}\right)^2}. \tag{4.21}$$

The coefficients S_u^* for all input neurons u are sorted in the descending order. After that, if a great fall in the sequence of S_u^* is found (i.e., $\frac{S_{u'}^*}{S_{u'+1}^*} < \beta$ for a given neuron u' and a constant parameter $0 < \beta < 1$), all neurons i in the input layer such that $S_i^* \leq S_{u'}^*$ are removed from the BP-network.

We took inspiration from the above described approach and proposed a simple pruning strategy applicable to both input and hidden neurons:

1. For each hidden and input layer L, we compute a pruning threshold δ_L in the following way:

$$\delta_L = \beta \underset{\{v,j,p\}}{\text{mean}} \left|S_{jv}^p\right| = \beta \underset{\{v,j,p\}}{\text{mean}} \left|\frac{\partial y_{pv}}{\partial y_{pj}}\right|, \tag{4.22}$$

   where p indexes the training patterns, v indexes the output neurons, j indexes the neurons in layer L, S_{jv}^p is the sensitivity of the output neuron v to the activity of neuron j for the input pattern p, and $0 < \beta \leq 1$ is a given constant parameter. In our experiments, β was set to 0.7.
2. If a neuron i in a hidden or input layer L fulfills the following condition:

$$\max_p \underset{v}{\text{mean}} \left|S_{iv}^p\right| < \delta_L, \tag{4.23}$$

   i is selected to be pruned.

3. All hidden and input neurons satisfying condition (4.23) are pruned at once.

The idea of our pruning criterion is, that an input or hidden neuron should be labeled as superfluous only if the corresponding sensitivity coefficients are low for all the training patterns. Our pruning strategy based on sensitivity analysis is summarized in Step 3. of Algorithm 4.4 on page 90.

3. How to control the pruning process?

In the following paragraphs, we will describe our training-and-pruning methodology [83, 85]. Although the methodology is independent of the chosen training algorithm, we hope it will be advantageous especially together with the SCGIR-algorithm and its enhancements.

Let $T = \{(\vec{x}_p, \vec{d}_p) \,|\, \vec{x}_p \in \Re^n, \vec{d}_p \in \Re^m, p \in \{1, \ldots, P\}\}$ be the training set and $V_{pr} = \{\,(\vec{x}_r, \vec{d}_r\,) \,|\, \vec{x}_r \in \Re^n, \vec{d}_r \in \Re^m, r \in \{1, \ldots, R\}\}$ and let $V_{es} = \{\,(\vec{x}_q, \vec{d}_q\,) \,|\, \vec{x}_q \in \Re^n, \vec{d}_q \in \Re^m, q \in \{1, \ldots, Q\}\}$ be two mutually disjoint validation sets. We denote the vectors $\vec{x}$ as the input patterns, and the vectors $\vec{d}$ as the desired network outputs. To control both the training and pruning processes together, we use a parameter *maxEpochs* and the validation sets V_{pr} and V_{es}. The training process with pruning can be briefly described by the following schema:

1. Initialize the BP-network.
2. Repeat the following two steps:
   (a) *Training:*
   (Re)train the BP-network on the training set T with early stopping and with the following stop criteria:
   - the overall error on the validation set V_{es} grows five times in a row,
   - or the maximum number of training epochs indicated by parameter *maxEpochs* is reached.

   Save the BP-network.
   (b) *Pruning:*
   i. Prune from the BP-network hidden neurons based on internal representation.
   ii. Prune from the BP-network hidden and input neurons based on sensitivity analysis.
   iii. Adjust input patterns from the training set T and the validation sets V_{es} and V_{pr}.

   Until no further pruning is possible (i.e., no input or hidden neurons were pruned during the last pruning phase).
3. Select the BP-network with the lowest performance error on the validation set V_{pr} to be the final one.

Now, we will describe the key aspects of the above specified methodology in detail. The parameter *maxEpochs* denotes the maximum total number of training epochs over all phases of (re)training. After each training phase, the parameter *maxEpochs* is reduced by the number of training epochs actually

used by the training algorithm. If *maxEpochs* falls to zero, the following training phases are implicitly skipped.

After each pruning phase t, the training set T and the validation sets V_{es} and V_{pr} are adjusted by removing from the input patterns all features corresponding to the input neurons actually pruned from the BP-network. For a BP-network M_t with the given n_t inputs, its training set T_t will have the form $T_t = \{(\vec{x}_p^t, \vec{d}_p) \mid \vec{x}_p^t \in \Re^{n_t}, \vec{d}_p \in \Re^m, p \in \{1, \ldots, P\}\}$, where the input patterns $\vec{x}_p^t$ contain exactly the features corresponding to the inputs of M_t. Analogically, $V_{pr,t} = \{ (\vec{x}_r^t, \vec{d}_r) \mid \vec{x}_r^t \in \Re^{n_t}, \vec{d}_r \in \Re^m, r \in \{1, \ldots, R\}\}$ and $V_{es,t} = \{ (\vec{x}_q^t, \vec{d}_q) \mid \vec{x}_q^t \in \Re^{n_t}, \vec{d}_q \in \Re^m, q \in \{1, \ldots, Q\}\}$.

The validation set $V_{pr,t}$ is used to measure the performance error of the corresponding BP-network M_t. For a BP-network M_t with the weight vector $\vec{w}_t$ and n_t inputs, the performance error $e(V_{pr,t})_t$ of M_t on $V_{pr,t}$ is computed using the following formula:

$$e(V_{pr,t})_t = E^{V_{pr,t}}(\vec{w}_t) = \frac{1}{2} \sum_r \| \vec{y}_r^t - \vec{d}_r \|^2 = \frac{1}{2} \sum_r \sum_v (y_{rv}^t - d_{rv})^2, \quad (4.24)$$

where r indexes all patterns from $V_{pr,t}$, v indexes all output neurons of M_t, and $\vec{y}_r^t \in \Re^m$ denotes the actual output produced by M_t for the input pattern $\vec{x}_r^t$. After each iteration t of the algorithm, we save the actual BP-network M_t and its performance $e(V_{pr,t})_t$. At the end, the BP-network with lowest value of $e(V_{pr})_t$ is selected to be the final one.

The validation sets $V_{es,t}$ are used by the early stopping strategy (see Section 3.4.1 for details). The training in each training phase t is stopped, if the overall error of the actual BP-network M_t on the corresponding validation set $V_{es,t}$ grows five times in a row or if the maximal number of training epochs (indicated by the parameter *maxEpochs*) is reached. The overall error function is specific to the chosen training algorithm (e.g., E defined by (1.8) for the standard BP-algorithm or the SCG-algorithm, or G defined by (4.2) for the SCGIR-algorithm).

Our training-and-pruning methodology [83, 85] is summarized by Algorithms 4.3 and 4.4 on pages 89 and 90.

4.2.4 Analytical sensitivity control

In the previous subsections, we introduced a methodology for training of BP-networks with pruning of hidden and input neurons based on internal representation and sensitivity analysis [83, 85] (described by Algorithms 4.3 and 4.4). The methodology can be combined e.g., with the formerly proposed SCGIR-training algorithm (described by Algorithm 4.1 on page 79). However, a sophisticated enhancement of the training method could make the pruning process much easier. It might also increase the chance that the trained and pruned BP-networks would have an (sub)optimal and transparent structure and that they would generalize adequately.

Especially the pruning process based on sensitivity analysis could be further simplified and improved by a well-designed and sophisticated regularization technique. However, most of the existing regularization techniques for structure optimization (e.g., weight decay [119], weight elimination [117]) are based on the minimization of the magnitudes of weights. Weights smaller than a given

Algorithm 4.3 Function *train_and_prune*() (training-and-pruning methodology based on internal representation and sensitivity analysis)

1. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of size W.
   A training algorithm ALG.
   Three mutually disjoint data sets:
   - a training set $T = \{(\vec{x}_p, \vec{d}_p) \,|\, \vec{x}_p \in \Re^n, \vec{d}_p \in \Re^m, p \in \{1, \ldots, P\}\}$,
   - a validation set for early stopping $V_{es} = \{(\vec{x}_q, \vec{d}_q) \,|\, \vec{x}_q \in \Re^n, \vec{d}_q \in \Re^m, q \in \{1, \ldots, Q\}\}$,
   - a validation set for pruning $V_{pr} = \{(\vec{x}_r, \vec{d}_r) \,|\, \vec{x}_r \in \Re^n, \vec{d}_r \in \Re^m, r \in \{1, \ldots, R\}\}$.

   The maximum number of training epochs $maxEpochs$, $maxEpochs \in N$.
   The pruning parameter β, $0 < \beta \leq 1$.
2. *Initialization*:
   Set the discrete time variable $t = 1$.
   Set $M_1 = M$, $T_1 = T$, $V_{es,1} = V_{es}$, $V_{pr,1} = V_{pr}$.
3. Repeat:
   (a) *Training / retraining:*
      i. Train the BP-network M_t on the training set T_t using the training algorithm ALG with early stopping (see Section 3.4.1) with the validation set $V_{es,t}$ and the maximum number of training epochs $maxEpochs$. Apply the following stop criteria on ALG:
         - The overall error on the validation set $V_{es,t}$ grows five times in a row.
         - The maximum number of training epochs is reached.

      ii. *Adjust the parameter maxEpochs*:
         Let $epochs_t \leq maxEpochs$ be the number of epochs used by the ALG-algorithm. Then:
         $maxEpochs = maxEpochs - epochs_t$.

      iii. *Performance evaluation:*
         Compute the performance error $e(V_{pr,t})_t$ of the BP-network M_t on the validation set $V_{pr,t}$:
         $e(V_{pr,t})_t = \frac{1}{2} \sum_r \sum_v (y_{rv} - d_{rv})^2$,
         where y_{rv} denotes the v-th actual output produced by M_t for the r-th input pattern from $V_{pr,t}$.

   (b) *Pruning:*
      i. Identify and remove from the BP-network the redundant hidden and input neurons (Algorithm 4.4):
         $M_{t+1} = prune\_hidden\_and\_input\_neurons(M_t, \beta)$.

      ii. Adjust the sets T_t, $V_{es,t}$ and $V_{pr,t}$: remove from the input patterns all features corresponding to the input neurons pruned from M_t.

   (c) Set $t = t + 1$.

   Until the following *stop criterion for pruning* is satisfied:
   - M_t and M_{t-1} have equal topologies (i.e., no input or hidden neurons were pruned during the t-th iteration of pruning).
4. Select the network $M_{t'}$ with the lowest error $e(V_{pr,t'})_{t'} = \min_{i \in \{1,\ldots,t\}} e(V_{pr,i})_i$ to be the final one.

Algorithm 4.4 Function *prune_hidden_and_input_neurons*() (one iteration of the pruning method)

1. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of size W.
   The pruning parameter β , $0 < \beta \leq 1$.
2. *Remove redundant hidden neurons based on the formed internal representation:* For each hidden layer L_k , $1 < k \leq l + 1$:
   (a) For each neuron i in layer L_k:
   If i forms a uniform representation (there exists $c \in \Re$ such that for all training patterns p: $y_{pi} = c$):
      i. Remove i and the corresponding edges from the BP-network M.
      ii. For all $j \in L_{k+1}$: set $w_{0j}^{new} = w_{0j} + c\, w_{ij}$.
   (b) For each pair of neurons i_1 and i_2 in layer L_k:
   If i_1 and i_2 form identical representations to each other ($y_{pi_1} = y_{pi_2}$ for all training patterns p):
      i. Remove i_2 and the corresponding edges from the BP-network M.
      ii. For all $j \in L_{k+1}$: set $w_{i_1 j}^{new} = w_{i_1 j} + w_{i_2 j}$.
   If i_1 and i_2 form complementary representations to each other ($y_{pi_1} = -y_{pi_2}$ for all training patterns p):
      i. Remove i_2 and the corresponding edges from the BP-network M.
      ii. For all $j \in L_{k+1}$: set $w_{i_1 j}^{new} = w_{i_1 j} - w_{i_2 j}$.
3. *Remove redundant hidden and input neurons based on sensitivity analysis:*
   (a) *Relevance evaluation:* For each hidden and input layer L (in the direction from the output to the input layer) and for each neuron i in layer L, compute the values of the sensitivity coefficients S_{iv}^p:
   $$S_{iv}^p = \begin{cases} f'(\xi_{pv})\, w_{iv} \,; & i \in L_{l+1} \\ \sum_g S_{gv}^p f'(\xi_{pg}) w_{ig} \,; & i \in L_k \,, g \in L_{k+1} \,, \, 1 \leq k \leq l+1 \,, \end{cases}$$
   for all output neurons (indexed by v) and all training patterns (indexed by p).
   (b) *Identification of redundant hidden and input neurons:*
   Set $I = \{\}$.
   For each hidden and input layer L:
      i. Compute the threshold δ_L:
      $$\delta_L = \beta \, \mathrm{mean}_{\{v,i,p\}} \, |S_{iv}^p| \,,$$
      where p indexes the training pattern, v indexes the output neurons, i indexes the neurons in layer L.
      ii. For each neuron i in layer L:
      If i fulfills the following condition:
      $\max_p \mathrm{mean}_v \, |S_{iv}^p| < \delta_L$,
      insert i into I.
   (c) *Remove redundant neurons:*
   Remove all the neurons $i \in I$ and the corresponding edges from the BP-network M.
4. *Return:* The pruned BP-network M.

threshold are regarded as useless and removed. A question is, whether such regularization techniques would also make easier the pruning of hidden and input neurons.

A problem of the regularization techniques based on weights' minimization is, that they favor configurations with many small weights to configurations with a few large weights. However, a BP-network with the configuration of the second type may perform and generalize better [95] and its internal structure may be simpler and more transparent. Moreover, regularization based on weights' minimization could prevent the BP-network from forming a condensed (or even bipolar) internal representation and make thus the pruning of hidden neurons based on their representations more difficult. Also the impact of weight minimization on the sensitivity based pruning is questionable.

For these reasons, rather than minimizing the magnitudes of weights, we decided to minimize directly the absolute sensitivity coefficients, which are a key part of our pruning strategy. In [83, 84, 85], we proposed a new regularization technique SC for analytical sensitivity control. The criterion for sensitivity control is formulated as a penalty term added to the error function to be minimized during training – the sensitivity error function.

In essence, there are two main options for sensitivity control in BP-networks – sensitivity can be either inhibited or enforced [79, 83]. Both variants are assumed to increase the differences among the achieved sensitivity coefficients of the respective neurons and restrict the space of candidate hypotheses for the wanted network function. Anyway, we expect that BP-networks with an inhibited sensitivity might yield better results due to their smoother function.

We call the SC-regularization technique analytical, because it works exactly with the sensitivity coefficients (defined by Equations (4.16) and (4.17)) and it doesn't attempt to approximate them in some way. Such regularization could make the training process computationally expensive. On the other side, it is maximally compatible with the proposed sensitivity-based pruning rule (Equation (4.23)). Thus the analytical SC-regularization might simplify and improve the pruning process more than a less precise yet more efficient regularization technique.

In sum, the SC-regularization technique is intended to impact pruning of more input and hidden neurons and easier optimization of the network structure. It should also contribute to smoother network functions and improved generalization ability of the trained model [16]. At the same time, the SC-regularization technique should improve (or at least not reduce) the ability of the trained BP-networks to form a condensed internal representation during training and support thus an easy interpretation of the extracted knowledge.

The sensitivity error function

In the following paragraphs, we will discuss the form of the sensitivity error function for the SC-method. In the case of sensitivity inhibition, the sensitivity error function should penalize input (and indirectly also hidden) neurons for high magnitudes of sensitivity coefficients S_{iv}^p (see Equations (4.16), (4.17) for the definition of S_{iv}^p). This would help the proposed pruning technique (Algorithm 4.4 on page 90) to identify and prune from the BP-network hidden and input neurons with low influence on the network outputs. Such redundant hidden and

input neurons are characterized by absolutely low sensitivity coefficients for all training patterns and network outputs.

In general, the sensitivity error function G will be formulated as it follows [83]:

$$G = G(\vec{w}) = \sum_p G_p(\vec{w}) = \frac{1}{2} \sum_p \sum_u \sum_v (S_{uv}^p)^2 = \frac{1}{2} \sum_p \sum_u \sum_v \left(\frac{\partial y_{pv}}{\partial x_{pu}} \right)^2 , \tag{4.25}$$

where p indexes the training patterns, v indexes the output neurons, u indexes the input neurons. For a given training pattern p, the term $\frac{\partial y_{pv}}{\partial x_{pu}}$ corresponds to the sensitivity S_{uv}^p of the activity y_{pv} of the output neuron v to the u-th input feature x_{pu}. $G_p(\vec{w}) = \frac{1}{2} \sum_{u,v} (S_{uv}^p)^2$ is the sensitivity error function corresponding to the p-th training pattern.

During the training process, the above-defined sensitivity error function G is optimized simultaneously with the performance error function E (defined by Equation (1.8)) and the representation error function F (defined by Equation (4.1)). We will thus use the following modification of the overall error function H:

$$H(\vec{w}) = E(\vec{w}) + c_F F(\vec{w}) + c_G G(\vec{w}) , \tag{4.26}$$

where $c_F \geq 0$ and $c_G \in \Re$ are coefficients reflecting the trade-off between the influence of E, F and G in H. The parameters c_F and c_G remain constant throughout the whole training process but might be variable, too. The parameter c_G controls, whether the sensitivity is inhibited or enforced during training. For sensitivity inhibition, we must set $c_G > 0$. For sensitivity enforcement (see, e.g., [79, 83]), it is necessary to set $c_G < 0$. In [79], sensitivity control is used for improved information transfer in modular networks.

In the following subsections, we will derive the rules for weight adjustment corresponding to the overall error function $H(\vec{w})$.

The derivation of the rules for weight adjustment

Because of the complexity of the error function $G(\vec{w})$ defined by Equation (4.25), we firstly derived in [85] the rules for weight adjustment of the SC-regularization technique for the simplified BP-network topology with just one hidden layer. In [84], we generalized the SC-technique to the the model of 'bipolar' BP-networks with an arbitrary number of hidden layers.

In the equations in this subsection, we will use the following notation: p will be an index over all training patterns, v will index all output neurons and u will index all input neurons. A weight from a neuron i to a neuron j will be denoted by w_{ij}, y will denote the activity of a neuron while ξ will be its potential. x will correspond to the considered element of the presented input pattern. For a given training pattern p, the term $\partial y_{pj} / \partial y_{pi}$ will correspond to the sensitivity S_{ij}^p of the activity y_{pj} of the (hidden or output) neuron j to the activity y_{pj} of the neuron i in one of the preceding layers. For the input neurons u, their activity is equal to their input (i.e., $y_{pu} = x_{pu}$).

We will work with the general form of the sensitivity error function G (defined by (4.25)). The overall error function $H(\vec{w}) = E(\vec{w}) + c_F F(\vec{w}) + c_G G(\vec{w})$ will be

evaluated according to:

$$H(\vec{w}) = \frac{1}{2}\sum_{p}\sum_{v}(\,y_{pv} - d_{pv}\,)^2 \;+\; c_F\sum_{p}\sum_{j'}(1+y_{pj'})^s\,(1-y_{pj'})^s\,y_{pj'}^2 + \\ + c_G\,\frac{1}{2}\sum_{p}\sum_{u}\sum_{v}(S_{uv}^p)^2\;. \tag{4.27}$$

In the above expression, j' indexes all hidden neurons, d denotes the desired activity of a neuron and s is a parameter for tuning the shape of the representation error function.

To minimize $H(\vec{w})$, the functions $E(\vec{w})$, $c_F F(\vec{w})$ and $c_G G(\vec{w})$ are minimized simultaneously. The derivative of $H(\vec{w})$ is given by $H'(\vec{w}) = E'(\vec{w}) + c_F F'(\vec{w}) + c_G G'(\vec{w})$. It is computed as $H'(\vec{w}) = \sum_p H'_p(\vec{w})$, where $H'_p(\vec{w}) = (..., \frac{\partial H_p}{\partial w_{ij}}, ...)$ and

$$\frac{\partial H_p}{\partial w_{ij}} \;=\; \frac{\partial E_p}{\partial w_{ij}} + c_F\frac{\partial F_p}{\partial w_{ij}} + c_G\frac{\partial G_p}{\partial w_{ij}}\,. \tag{4.28}$$

The partial derivatives in $\partial E_p/\partial w_{ij} + c_F\partial F_p/\partial w_{ij}$ were already stated in Equation (4.15). To inhibit also the sensitivity of the network, we have to determine the terms $\partial G_p/\partial w_{ij}$ as well.

Let $(\vec{x}_p, \vec{d}_p)$ be the training pattern presented to the BP-network in time t and let i be the neuron connected with neuron j via the weight w_{ij}. In the derivations below, all neurons in the same layer as neuron i will be indexed by g, all neurons in the same layer as neuron j will be indexed by h.

The derivative $\frac{\partial G_p}{\partial w_{ij}}$ we can be computed using the sum and chain rules:

$$\frac{\partial G_p}{\partial w_{ij}} \;=\; \frac{\partial}{\partial w_{ij}}\left(\frac{1}{2}\sum_{u}\sum_{v}(S_{uv}^p)^2\right) \;=\; \sum_{u}\sum_{v}S_{uv}^p\left(\frac{\partial S_{uv}^p}{\partial w_{ij}}\right)\,. \tag{4.29}$$

- For neuron j in the output layer L_{l+2}:
  The term S_{uv}^p can be formulated as:

$$S_{uv}^p \;=\; \sum_{g} S_{gv}^p\,S_{ug}^p \;=\; \sum_{g} w_{gv}\,S_{ug}^p\,, \tag{4.30}$$

  where g indexes neurons in the last hidden layer L_{l+1}. Therefore,

$$\frac{\partial G_p}{\partial w_{ij}} = \sum_{u}\sum_{v}S_{uv}^p\frac{\partial}{\partial w_{ij}}\left(\sum_{g} w_{gv}\,S_{ug}^p\right) \;=\; \sum_{u}\sum_{v}S_{uv}^p\sum_{g}S_{ug}^p\frac{\partial w_{gv}}{\partial w_{ij}}\,. \tag{4.31}$$

  Because

$$\frac{\partial w_{gv}}{\partial w_{ij}} \;=\; \begin{cases} 1 & \text{for } (g=i) \text{ and } (v=j) \\ 0 & \text{otherwise }, \end{cases} \tag{4.32}$$

  we obtain:

$$\frac{\partial G_p}{\partial w_{ij}} \;=\; \sum_{u} S_{uj}^p S_{ui}^p\,. \tag{4.33}$$

- For neuron j in a hidden layer other than first ($j \in L_k$, $i \in L_{k-1}$, $2 < k \leq l+1$):
  The term S^p_{uv} can be formulated as:

$$S^p_{uv} = \sum_g S^p_{ug} S^p_{gv} = \sum_g S^p_{ug} \left(\sum_h S^p_{gh} S^p_{hv} \right) = \\ = \sum_g S^p_{ug} \sum_h (1 - y^2_{ph}) w_{gh} S^p_{hv} \,, \tag{4.34}$$

  where g indexes all neurons in the layer L_{k-1}, h indexes all neurons in the layer L_k. Therefore, we can apply the sum and product rules and obtain:

$$\frac{\partial G_p}{\partial w_{ij}} = \sum_u \sum_v S^p_{uv} \frac{\partial}{\partial w_{ij}} \left(\sum_g S^p_{ug} \sum_h (1 - y^2_{ph}) w_{gh} S^p_{hv} \right) = \\ = \sum_u \sum_v S^p_{uv} \sum_g S^p_{ug} \sum_h \frac{\partial \left[(1 - y^2_{ph}) w_{gh} S^p_{hv} \right]}{\partial w_{ij}} = \\ = \sum_u \sum_v S^p_{uv} \sum_g S^p_{ug} \sum_h \left[\frac{\partial (1 - y^2_{ph})}{\partial w_{ij}} w_{gh} S^p_{hv} + \right. \\ \left. + (1 - y^2_{ph}) \frac{\partial w_{gh}}{\partial w_{ij}} S^p_{hv} + (1 - y^2_{ph}) w_{gh} \frac{\partial S^p_{hv}}{\partial w_{ij}} \right] . \tag{4.35}$$

  The particular derivatives will be computed in the following way: $\frac{\partial (1 - y^2_{ph})}{\partial w_{ij}} = 0$ for $h \neq j$ and

$$\frac{\partial (1 - y^2_{pj})}{\partial w_{ij}} = \frac{\partial (1 - y^2_{pj})}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial \xi_{pj}} \frac{\partial \xi_{pj}}{\partial w_{ij}} = (-2\, y_{pj})\, (1 - y^2_{pj})\, x_{pi} \,. \tag{4.36}$$

  The derivative $\frac{\partial w_{gh}}{\partial w_{ij}}$ can be computed analogically to the Equation (4.32) and

$$\frac{\partial S^p_{hv}}{\partial w_{ij}} = \frac{\partial S^p_{hv}}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial \xi_{pj}} \frac{\partial \xi_{pj}}{\partial w_{ij}} = T^p_{hv,j}\, (1 - y^2_{pj})\, y_{pi} \,, \tag{4.37}$$

  where $T^p_{hv,j}$ denotes an auxiliary term, $T^p_{hv,j} = \frac{\partial S^p_{hv}}{\partial y_{pj}}$.
  To obtain the value of $T^p_{hv,j}$, the term $T^p_{qv,j} = \frac{\partial S^p_{qv}}{\partial y_{pj}}$ has to be computed recursively backwards from the output layer towards the hidden layer L_k containing the neurons h and j.

  - For a neuron q in the last hidden layer L_{l+1}:

$$T^p_{qv,j} = \frac{\partial S^p_{qv}}{\partial y_{pj}} = \frac{\partial w_{qv}}{\partial y_{pj}} = 0. \tag{4.38}$$

– For a neuron q in the hidden layer $L_{k'}$, $k \leq k' \leq l$:

$$
\begin{aligned}
T^p_{qv,j} = \frac{\partial}{\partial y_{pj}} \left(\sum_r S^p_{qr} S^p_{rv} \right) &= \frac{\partial}{\partial y_{pj}} \left(\sum_r w_{qr} \left(1 - y^2_{pr}\right) S^p_{rv} \right) = \\
&= \sum_r w_{qr} \frac{\partial \left[(1 - y^2_{pr}) S^p_{rv} \right]}{\partial y_{pj}} = \\
&= \sum_r w_{qr} \left[\frac{\partial (1 - y^2_{pr})}{\partial y_{pj}} S^p_{rv} + (1 - y^2_{pr}) \frac{\partial S^p_{rv}}{\partial y_{pj}} \right] = \\
&= \sum_r w_{qr} \left[\frac{\partial (1 - y^2_{pr})}{\partial y_{pr}} \frac{\partial y_{pr}}{\partial y_{pj}} S^p_{rv} + (1 - y^2_{pr}) T^p_{rv,j} \right] = \\
&= \sum_r w_{qr} \left[\left(-2 y_{pr}\right) S^p_{jr} S^p_{rv} + (1 - y^2_{pr}) T^p_{rv,j} \right] ,
\end{aligned}
\tag{4.39}
$$

where r indexes neurons from the hidden layer $L_{k'+1}$.

Altogether:

$$
\begin{aligned}
\frac{\partial G_p}{\partial w_{ij}} = \sum_u \sum_v S^p_{uv} \Bigg[& \sum_g S^p_{ug} (-2\, y_{pj}) \left(1 - y^2_{pj}\right) y_{pi}\, w_{gj}\, S^p_{jv} + \\
& + S^p_{ui} \left(1 - y^2_{pj}\right) S^p_{jv} + \\
& + \sum_g S^p_{ug} \sum_h \left(1 - y^2_{ph}\right) w_{gh}\, T^p_{hv,j} \left(1 - y^2_{pj}\right) y_{pi} \Bigg] = \\
= (-2\, y_{pj})\, y_{pi} & \sum_u \sum_v S^p_{uv}\, S^p_{jv} \sum_g S^p_{ug}\, w_{gj} \left(1 - y^2_{pj}\right) + \\
& + \left(1 - y^2_{pj}\right) \sum_u \sum_v S^p_{uv}\, S^p_{ui}\, S^p_{jv} + \\
& + \left(1 - y^2_{pj}\right) y_{pi} \sum_u \sum_v S^p_{uv} \sum_g S^p_{ug} \sum_h \left(1 - y^2_{ph}\right) w_{gh}\, T^p_{hv,j} .
\end{aligned}
\tag{4.40}
$$

Because for the first subterm,

$$
\sum_g S^p_{ug}\, w_{gj} \left(1 - y^2_{pj}\right) = \sum_g S^p_{ug}\, S^p_{gj} = S^p_{uj} , \tag{4.41}
$$

we obtain:

$$
\begin{aligned}
\frac{\partial G_p}{\partial w_{ij}} = (-2\, y_{pj})\, y_{pi} & \sum_u \sum_v S^p_{uv}\, S^p_{jv}\, S^p_{uj} + \\
& + \left(1 - y^2_{pj}\right) \sum_u \sum_v S^p_{uv}\, S^p_{ui}\, S^p_{jv} + \\
& + \left(1 - y^2_{pj}\right) y_{pi} \sum_u \sum_v S^p_{uv} \sum_g S^p_{ug} \sum_h \left(1 - y^2_{ph}\right) w_{gh}\, T^p_{hv,j} .
\end{aligned}
\tag{4.42}
$$

- For neuron j in the first hidden layer L_2 (neuron i is in the input layer L_1): The term S^p_{uv} can be formulated as:

$$S^p_{uv} \;=\; \sum_h S^p_{uh} S^p_{hv} \;=\; \sum_h (1 - y^2_{ph}) w_{uh}\, S^p_{hv}\,, \tag{4.43}$$

where h indexes all neurons in the first hidden layer.
Therefore, we can apply the sum and product rules and obtain:

$$\begin{aligned}
\frac{\partial G_p}{\partial w_{ij}} &= \sum_u \sum_v S^p_{uv} \frac{\partial}{\partial w_{ij}} \left(\sum_h (1 - y^2_{ph}) w_{uh}\, S^p_{hv} \right) = \\
&= \sum_u \sum_v S^p_{uv} \sum_h \frac{\partial \left[(1 - y^2_{ph}) w_{uh}\, S^p_{hv} \right]}{\partial w_{ij}} = \\
&= \sum_u \sum_v S^p_{uv} \sum_h \left[\frac{\partial (1 - y^2_{ph})}{\partial w_{ij}}\, w_{uh}\, S^p_{hv} + (1 - y^2_{ph}) \frac{\partial w_{uh}}{\partial w_{ij}}\, S^p_{hv} + \right. \\
&\qquad \left. + (1 - y^2_{ph})\, w_{uh} \frac{\partial S^p_{hv}}{\partial w_{ij}} \right] .
\end{aligned} \tag{4.44}$$

The particular derivatives can be computed analogically to the previous case. Altogether:

$$\begin{aligned}
\frac{\partial G_p}{\partial w_{ij}} &= \sum_u \sum_v S^p_{uv}\, (-2\, y_{pj})\, (1 - y^2_{pj})\, x_{pi}\, w_{uj}\, S^p_{jv} + \\
&\quad + \sum_v S^p_{iv}\, (1 - y^2_{pj})\, S^p_{jv} + \\
&\quad + \sum_u \sum_v S^p_{uv} \sum_h (1 - y^2_{ph})\, w_{uh}\, T^p_{hv,j}\, (1 - y^2_{pj})\, x_{pi} \;= \\
&= (-2\, y_{pj})\, x_{pi} \sum_u \sum_v S^p_{uv}\, S^p_{uj}\, S^p_{jv} + \\
&\quad + (1 - y^2_{pj}) \sum_v S^p_{iv}\, S^p_{jv} + \\
&\quad + (1 - y^2_{pj})\, x_{pi} \sum_u \sum_v S^p_{uv} \sum_h (1 - y^2_{ph})\, w_{uh}\, T^p_{hv,j}\,.
\end{aligned} \tag{4.45}$$

The SCGS-algorithm The second version of our framework for training of the BP-networks will be called SCGS [83, 84, 85]. It combines the SC- and IR-regularization techniques with the SCG-algorithm (described by Algorithm 4.2 on page 80) and with the above-defined training-and-pruning methodology (Algorithms 4.3 and 4.4 on pages 89 and 90).

The training process of the SCGS-method is identical to the standard SCG-algorithm (see Algorithm 4.2 on page 80 for the general schema of the SCG-like algorithms), however with $H(\vec{w}) = E(\vec{w}) + c_F F(\vec{w}) + c_G G(\vec{w})$ (defined by (4.26)) as the error function. The derivative of $H(\vec{w})$ is given by $H'(\vec{w}) = E'(\vec{w}) + c_F F'(\vec{w}) + c_G G'(\vec{w})$. It is computed as: $H'(\vec{w}) = \sum_p H'_p(\vec{w})$, where $H'_p(\vec{w}) = (..., \frac{\partial H_p}{\partial w_{ij}}, ...)$ and $\partial H_p / \partial w_{ij} = \frac{\partial E_p}{\partial w_{ij}} + c_F \frac{\partial F_p}{\partial w_{ij}} + c_G \frac{\partial G_p}{\partial w_{ij}}$.

For the considered 'bipolar' BP-network model with an arbitrary number of hidden layers, the terms $\partial H_p/\partial w_{ij}$ correspond to:

$$\frac{\partial H_p}{\partial w_{ij}} = \begin{cases} (y_{pj} - d_{pj}) y_{pi} + c_G \sum_u S^p_{uj} S^p_{ui} \\ \qquad\qquad \text{for neuron } j \in L_{l+2} \\ \Big\{ \sum_q \delta_{pq} w_{jq} + 2\, c_F \left[1 - (s+1) y^2_{pj}\right] (1 - y^2_{pj})^{s-1} y_{pj} \Big\} (1 - y^2_{pj})\, y_{pi} + \\ \quad + c_G \Big[(-2 y_{pj}) y_{pi} \sum_{u,v} S^p_{uv} S^p_{jv} S^p_{uj} + \\ \quad + (1 - y^2_{pj}) \sum_{u,v} S^p_{uv} S^p_{ui} S^p_{jv} + \\ \quad + (1 - y^2_{pj}) y_{pi} \sum_{u,v} S^p_{uv} \sum_g S^p_{ug} \sum_h (1 - y^2_{ph}) w_{gh} T^p_{hv,j} \Big] \\ \qquad\qquad \text{for neuron } j \in L_{l+1} \\ \Big\{ \sum_q (\delta_{pq} + c_F \varrho_{pq})\, w_{jq} + 2\, c_F \left[1 - (s+1) y^2_{pj}\right] (1 - y^2_{pj})^{s-1} y_{pj} \Big\} \cdot \\ \quad \cdot (1 - y^2_{pj})\, y_{pi} + \\ \quad + c_G \Big[(-2 y_{pj}) y_{pi} \sum_{u,v} S^p_{uv} S^p_{jv} S^p_{uj} + \\ \quad + (1 - y^2_{pj}) \sum_{u,v} S^p_{uv} S^p_{ui} S^p_{jv} + \\ \quad + (1 - y^2_{pj}) y_{pi} \sum_{u,v} S^p_{uv} \sum_g S^p_{ug} \sum_h (1 - y^2_{ph}) w_{gh} T^p_{hv,j} \Big] \\ \qquad\qquad \text{for neuron } j \in L_k\,,\ 3 \le k \le l \\ \Big\{ \sum_q (\delta_{pq} + c_F \varrho_{pq})\, w_{jq} + 2\, c_F \left[1 - (s+1) y^2_{pj}\right] (1 - y^2_{pj})^{s-1} y_{pj} \Big\} \cdot \\ \quad \cdot (1 - y^2_{pj})\, y_{pi} + \\ \quad + c_G \Big[(-2 y_{pj}) x_{pi} \sum_{u,v} S^p_{uv} S^p_{uj} S^p_{jv} + (1 - y^2_{pj}) \sum_v S^p_{iv} S^p_{jv} + \\ \quad + (1 - y^2_{pj}) x_{pi} \sum_{u,v} S^p_{uv} \sum_h (1 - y^2_{ph}) w_{uh} T^p_{hv,j} \Big] \\ \qquad\qquad \text{for neuron } j \in L_2 \end{cases} \tag{4.46}$$

In the above expression, s denotes the parameter for tuning the shape of the representation error function, g, h index the neurons in the same layers as i and j, respectively, q is the index of the neurons in the layer above the neuron j. c_F and c_G are constants representing the influence of the respective error terms, δ_{pq} can be determined according to (1.18) and ϱ_{pq} according to (4.14). The terms $T^p_{hv,j} = \partial S^p_{hv}/\partial y_{pj}$ are computed recursively backwards from the output layer towards the hidden layer L_k containing the neuron j:

1. For each neuron h in the last hidden layer L_{l+1}:

$$T^p_{hv,j} = 0\,. \tag{4.47}$$

2. For each neuron h in the hidden layer $L_{k'}$, $k \le k' \le l$, $j \in L_k$:

$$T^p_{hv,j} = \sum_r w_{hr} \left[(-2 y_{pr})\, S^p_{jr} S^p_{rv} + (1 - y^2_{pr}) T^p_{rv,j} \right], \tag{4.48}$$

where r indexes neurons from the hidden layer above the neuron h.

To speed up the training process, all the terms S^p_{uv}, S^p_{uh} and S^p_{hv} (for all the training patterns p, hidden neurons h, output neurons v and input neurons u) are computed and saved before the computation of $H'(\vec{w})$ using the following schema:

- The values of sensitivity coefficients S^p_{uj} for all training patterns (indexed by p), all input neurons (indexed by u) and for all hidden and output neurons (denoted by j) are determined recursively in the direction from the first hidden layer towards the output layer:

  1. For a neuron j in the first hidden layer:

$$S^p_{uj} \;=\; \frac{\partial y_{pj}}{\partial x_{pi}} \;=\; (1 - y^2_{pj})\, w_{uj}\,. \tag{4.49}$$

  2. For a neuron j in a hidden layer other than first or in the output layer:

$$S^p_{uj} \;=\; \sum_g S^p_{gj}\, S^p_{ug} \;=\; (1 - y^2_{pj}) \sum_g w_{gj} S^p_{ug}\,, \tag{4.50}$$

     where g indexes neurons from the layer below the neuron j.

- Similarly, the values of sensitivity coefficients S^p_{iv} for all training patterns (indexed by p), all output neurons (indexed by v) and for all hidden neurons (denoted by i) are determined recursively in the direction from the last hidden layer towards the first hidden layer:

  1. For a neuron i in the last hidden layer:

$$S^p_{iv} = w_{iv}\,. \tag{4.51}$$

  2. For a neuron j in a hidden layer other than last:

$$S^p_{iv} \;=\; \sum_h S^p_{ih}\, S^p_{hv} \;=\; \sum_h w_{ih}\,(1 - y^2_{ph}) S^p_{hv} \tag{4.52}$$

     where h stands for the neurons from the layer above the neuron i.

The SCGS-method is summarized by Algorithm 4.5 on page 98.

Algorithm 4.5 Function $SCGS()$ (Scaled conjugate gradients algorithm with learning internal representation and analytical sensitivity control):

---

1. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of size W.
   A training set $T = \{\, (\vec{x}_p, \vec{d}_p\,) \mid \quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$.
2. *Train:*
   Train the BP-network M on the training set T using the SCG-training algorithm (described by Algorithm 4.2 on page 80) and the overall error function $H(\vec{w}) = \sum_p H_p(\vec{w})$ defined by (4.27) with $\frac{\partial H_p}{\partial w_{ij}}$ defined by (4.46).
   $M' = SCG(M, T, H)$.
3. *Return:* M'.

4.2.5 Summary of Section 4.2

In the first part of this section, we proposed a methodology for training of BP-networks with pruning of hidden and input neurons based on internal representation and sensitivity analysis (Algorithms 4.3 and 4.4). After that, we designed a new analytical sensitivity-based regularization technique SC. Finally, we combined the SC-regularization technique and the new-proposed training-and-pruning methodology with the formerly defined SCGIR-algorithm (see Algorithm 4.1 on page 79) – in the SCGS-algorithm (Algorithm 4.5).

The presented training-and-pruning methodology with the help of the proposed regularization techniques is intended to find an adequate and simple topology of the BP-network automatically during training. Moreover, it should identify relevant input features in the data and important hidden neurons. Both regularization techniques might also contribute to a smoother network function and better generalization ability of the trained BP-networks. As a result, the SCGS-algorithm should take advantage of all of the included techniques and provide a general framework capable of

1. analytical sensitivity control during training,
2. enforcement of condensed internal representation during training,
3. sophisticated pruning of hidden and input neurons and identification of significant input features,
4. improved generalization,
5. and easier interpretation of the extracted knowledge.

An expected drawback of the SCGS-method are its excessive time costs, which increase with the growing number of hidden layers. They are caused mainly by the complexity of the SC-regularization technique. The evaluation of the sensitivity error function and the corresponding rules for weight adjustment require time-consuming computation. Moreover, in each step of the training algorithm, we have to go through the whole network several times (e.g., to compute the values of the sensitivity coefficients and their derivatives).

The qualities and drawbacks of our training-and-pruning methodology and of the SCGS-method will be assessed in the experimental part of this work in Chapter 5. For more experimental results, see also [83, 84, 85].

4.3 Fast creation of a simple and clear internal structure

4.3.1 Introduction

The third goal – recalled Our third goal will be to enhance our framework to achieve both the first and the second goals at once:

- **Generalization:** The framework should favor smooth BP-network functions that contribute to adequate generalization.

- **Speed:** The training process should be as fast as possible and robust to the choice of initial parameters.

- **Creation of a simple and transparent network structure:** The BP-network should create a small, simple and transparent internal structure during training. It should also automatically identify relevant input features. The created model structure should simplify the further knowledge extraction.

In the previous Section 4.2, we proposed the SCGS-method for training of BP-networks (Algorithm 4.5 on page 98). It fulfills two of the above requirements – on creation of a simple and transparent network structure and on improved generalization. On the other hand, a problem of this method is its low efficiency.

The high time and space complexities of the SCGS-algorithm are caused mainly by the SC-regularization technique. The efficiency problem of the SC-method arises from the analytical nature of the sensitivity error function and from the complexity of the corresponding rules for weight adjustment. On the other hand, the SC-regularization technique is (thanks to its analytical character) highly precise and maximally compatible with our sensitivity-based pruning technique (Algorithm 4.4 on page 90).

Contrary to the SC-regularization technique, some other methods included in our framework are very efficient. They are especially the IR-regularization technique together with the SCG-training algorithm and our training-and-pruning methodology. For this reason, when proposing our approach to the third goal, we will keep these methods in our framework. However, we will introduce a new approximative sensitivity-based regularization technique that will substitute the analytical one.

The new approximative sensitivity-based regularization technique should be as efficient as possible. At the same time, it should also keep most of the good qualities of the analytical method:

1. it should simplify pruning based on sensitivity analysis,
2. it shouldn't be in conflict with the methods for improved transparency,
3. it should improve the generalization ability of the model,
4. and it should support creation of a smooth network function.

In the following paragraphs, we will describe our approach in detail.

4.3.2 Approximative sensitivity control

In this subsection, we will introduce our new approximative regularization technique for sensitivity control. We will call the method SCA [81]. In order to avoid the huge computational complexity of the analytical SC-technique, we will replace the sensitivity error function G (defined by (4.25)) by an alternative (approximative) expression. Naturally, there are more possible ways, in which G could be approximated. For this reason, we have decided to introduce several variants of the approximative sensitivity error function and select the most promising expression to be the final one.

We will propose the SCA-regularization technique for the 'bipolar' BP-network model, where all the hidden neurons have the hyperbolic tangent transfer functions, while all the output neurons implement the linear transfer functions. The approach could be applied also to BP-networks with other continuous and differentiable transfer functions, but the adaptation rules had to be slightly altered, reflecting the derivatives of the applied transfer functions.

In the equations in this subsection, we will use the following notation: p will be an index over all training patterns, v will index all output neurons, u will index all input neurons. A weight from a neuron i to a neuron j will be denoted by w_{ij}, y will denote the activity of a neuron while ξ will be its potential. x will correspond to the considered element of the presented input pattern.

In the following paragraphs, we will describe two possible expressions of the sensitivity error function and discuss their advantages and limitations.

The SCA1-method

We will denote the initial variant of the SCA-method as SCA1. It is based on a straightforward approximation of the overall network sensitivity. The exact sensitivity terms $S^p_{uv} = \frac{\partial y_{pv}}{\partial x_{pu}}$ (defined by Equations (4.16) and (4.17)) are approximated by the terms

$$T^p_{uv} = \frac{\Delta y_{pv}}{\Delta x_{pu}} = \frac{y_{pu} - y^*_{pu}}{x_{pu} - x^*_{pu}} \tag{4.53}$$

with $\Delta x_{pu} = x_{pu} - x^*_{pu}$ and $\Delta y_{pv} = y_{pv} - y^*_{pv}$. x^*_{pu} denotes the value of x_{pu} corrupted by a small amount of random noise (in or experiments usually chosen within the range of 0.01-1%). y^*_{pv} corresponds to the v-th actual network output determined for the noise-corrupted input pattern $\vec{x^*_p}$.

The sensitivity error function $G(\vec{w}) = \frac{1}{2}\sum_{p,u,v}\left(\frac{\partial y_{pv}}{\partial x_{pu}}\right)^2$ is replaced by the following alternative expression of the overall network sensitivity:

$$\begin{aligned} G^1 = G^1(\vec{w}) &= \sum_p G^1_p(\vec{w}) = \frac{1}{2}\sum_p\sum_u\sum_v (T^p_{uv})^2 = \\ &= \frac{1}{2}\sum_p\sum_u\sum_v \left(\frac{\Delta y_{pv}}{\Delta x_{pu}}\right)^2 = \frac{1}{2}\sum_p\sum_u\sum_v \frac{(y_{pv} - y^*_{pv})^2}{(x_{pu} - x^*_{pu})^2}\,, \end{aligned} \tag{4.54}$$

where $G^1_p(\vec{w}) = \frac{1}{2}\sum_{u,v}\left(\frac{\Delta y_{pv}}{\Delta x_{pu}}\right)^2$ is the approximative sensitivity error function corresponding to the p-th training pattern. The idea of this error criterion is, that the network outputs shouldn't be too sensitive to small changes of the inputs.

The overall error function is defined as $H(\vec{w}) = E(\vec{w}) + c_F F(\vec{w}) + c_G G^1(\vec{w})$. $E(\vec{w})$ represents here the performance error function (defined by Equation (1.8)), $F(\vec{w})$ stands for the representation error function (defined by Equation (4.1)) and $G^1(\vec{w})$ corresponds to the new-proposed network criterion minimized during training. The coefficients c_F and c_G reflect the trade-off between the influence of E, F and G^1 in H.

In sum, $H(\vec{w})$ has the form:

$$H(\vec{w}) = \frac{1}{2} \sum_p \sum_v (\, y_{pv} - d_{pv}\,)^2 \; + \; c_F \sum_p \sum_{j'} (1 + y_{pj'})^s \, (1 - y_{pj'})^s \, y_{pj'}^2 \; + \\ + \, c_G \, \frac{1}{2} \sum_p \sum_u \sum_v \left(\frac{\Delta y_{pv}}{\Delta x_{pu}} \right)^2 , \tag{4.55}$$

where j' indexes all hidden neurons, d denotes the desired activity of a neuron and s is a parameter for tuning the shape of the representation error function.

To minimize $H(\vec{w})$, the functions $E(\vec{w})$, $c_F F(\vec{w})$ and $c_G G^1(\vec{w})$ are minimized simultaneously. The derivative of $H(\vec{w})$ is given by $H'(\vec{w}) = E'(\vec{w}) + c_F F'(\vec{w}) + c_G G^{1\prime}(\vec{w})$. It is computed as: $H'(\vec{w}) = \sum_p H'_p(\vec{w})$, where $H'_p(\vec{w}) = (..., \frac{\partial H_p}{\partial w_{ij}}, ...)$ and

$$\frac{\partial H_p}{\partial w_{ij}} \; = \; \frac{\partial E_p}{\partial w_{ij}} + c_F \frac{\partial F_p}{\partial w_{ij}} + c_G \frac{\partial G_p^1}{\partial w_{ij}} \, . \tag{4.56}$$

The partial derivatives in $\partial E_p / \partial w_{ij} + c_F \partial F_p / \partial w_{ij}$ were already stated in Equation (4.15). To inhibit also the sensitivity of the network, we have to determine also the terms $\partial G_p^1 / \partial w_{ij}$.

The derivation of the rules for weight adjustment: Let $(\vec{x}_p, \vec{d}_p)$ be the training pattern presented to the BP-network in time t and $\vec{x}_p^*$ be the noise-corrupted input pattern.

In the following paragraphs, v will index all output neurons and u will index all input neurons. All neurons from the layer above the neuron j will be declared as those indexed by q. The activity and the potential of a neuron j corresponding to the input pattern $\vec{x}_p$ will be denoted as y_{pj} and ξ_{pj}, respectively. The activity and the potential of a neuron j corresponding to the noise-corrupted input pattern $\vec{x}_p^*$ will be denoted as y_{pj}^* and ξ_{pj}^*, respectively. For the input neurons u, their activity is equal to their input (i.e., $y_{pu} = x_{pu}$, $y_{pu}^* = x_{pu}^*$), $\Delta x_{pu} = x_{pu} - x_{pu}^*$. For the output neurons v, $\Delta y_{pv} = y_{pv} - y_{pv}^*$.

Let i be the neuron connected with neuron j via the weight w_{ij}. The terms for $\frac{\partial G_p^1}{\partial w_{ij}}$ can be computed using the sum and chain rules:

$$\frac{\partial G_p^1}{\partial w_{ij}} \; = \; \frac{\partial}{\partial w_{ij}} \left(\frac{1}{2} \sum_u \sum_v \frac{\Delta^2 y_{pv}}{\Delta^2 x_{pu}} \right) \; = \; \frac{1}{2} \left(\sum_u \frac{1}{\Delta^2 x_{pu}} \right) \sum_v \frac{\partial (\Delta^2 y_{pv})}{\partial w_{ij}} \, . \tag{4.57}$$

Because

$$\frac{\partial (\Delta^2 y_{pv})}{\partial w_{ij}} \; = \; 2 \, \Delta y_{pv} \frac{\partial (\Delta y_{pv})}{\partial w_{ij}} \; = \; 2 \, \Delta y_{pv} \left(\frac{\partial y_{pv}}{\partial w_{ij}} \; - \; \frac{\partial y_{pv}^*}{\partial w_{ij}} \right) , \tag{4.58}$$

we obtain:

$$\frac{\partial G_p^1}{\partial w_{ij}} = \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial w_{ij}} - \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial w_{ij}} \,. \quad (4.59)$$

Let

$$\begin{aligned} \sigma_{pj} &= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial \xi_{pj}} \,, \\ \sigma^*_{pj} &= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} \end{aligned} \quad (4.60)$$

be auxiliary terms for the back-propagated sensitivity errors. Because $\frac{\partial \xi_{pj}}{\partial w_{ij}} = y_{pi}$ and $\frac{\partial \xi^*_{pj}}{\partial w_{ij}} = y^*_{pi}$, we can apply the chain rule to (4.59) and get:

$$\begin{aligned} \frac{\partial G_p^1}{\partial w_{ij}} &= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial \xi_{pj}} \frac{\partial \xi_{pj}}{\partial w_{ij}} - \\ &\quad - \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} \frac{\partial \xi^*_{pj}}{\partial w_{ij}} \quad = \\ &= \sigma_{pj} y_{pi} - \sigma^*_{pj} y^*_{pi} \,. \end{aligned} \quad (4.61)$$

It remains to derive the terms for σ_{pj} and σ^*_{pj}.

- For neuron j in the output layer L_{l+2}:
  The derivative $\frac{\partial y_{pv}}{\partial \xi_{pj}}$ is given by:

$$\frac{\partial y_{pv}}{\partial \xi_{pj}} = \begin{cases} 0 & \text{for } j \neq v \\ f'(\xi_{pj}) & \text{for } j = v \,. \end{cases} \quad (4.62)$$

  Similarly, the derivative $\frac{\partial y^*_{pv}}{\partial \xi^*_{pj}}$ will be computed as:

$$\frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} = \begin{cases} 0 & \text{for } j \neq v \\ f'(\xi^*_{pj}) & \text{for } j = v \,. \end{cases} \quad (4.63)$$

  For the considered 'bipolar' BP-network model and the output neuron j, $f'(\xi_{pj}) = f'(\xi^*_{pj}) = 1$.

  Therefore, the terms σ_{pj} and σ^*_{pj} will be computed as it follows:

$$\sigma_{pj} \;=\; \sigma^*_{pj} \;=\; \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \Delta y_{pj} \,. \quad (4.64)$$

- For neuron j in a hidden layer ($j \in L_k$, $i \in L_{k-1}$, $1 < k \leq l+1$):
  The terms $\frac{\partial y_{pv}}{\partial \xi_{pj}}$ and $\frac{\partial y^*_{pv}}{\partial \xi^*_{pj}}$ will be computed indirectly using the chain rule:

$$\frac{\partial y_{pv}}{\partial \xi_{pj}} = \sum_q \frac{\partial y_{pv}}{\partial \xi_{pq}} \frac{\partial \xi_{pq}}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial \xi_{pj}} \;=\; \sum_q \frac{\partial y_{pv}}{\partial \xi_{pq}} w_{jq} f'(\xi_{pj}) \,, \quad (4.65)$$

$$\frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} = \sum_q \frac{\partial y^*_{pv}}{\partial \xi^*_{pq}} \frac{\partial \xi^*_{pq}}{\partial y^*_{pj}} \frac{\partial y^*_{pj}}{\partial \xi^*_{pj}} \;=\; \sum_q \frac{\partial y^*_{pv}}{\partial \xi^*_{pq}} w_{jq} f'(\xi^*_{pj})\,, \tag{4.66}$$

where q indexes all neurons in the layer L_{k+1}.

Therefore:

$$\begin{aligned}
\sigma_{pj} &= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial \xi_{pj}} \;= \\
&= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \left[\sum_q \frac{\partial y_{pv}}{\partial \xi_{pq}} w_{jq} f'(\xi_{pj})\right] \;= \\
&= f'(\xi_{pj}) \sum_q w_{jq} \left[\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial \xi_{pq}}\right] \;= \\
&= f'(\xi_{pj}) \sum_q w_{jq} \sigma_{pq}\,,
\end{aligned} \tag{4.67}$$

and

$$\begin{aligned}
\sigma^*_{pj} &= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} \;= \\
&= \left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \left[\sum_q \frac{\partial y^*_{pv}}{\partial \xi^*_{pq}} w_{jq} f'(\xi^*_{pj})\right] \;= \\
&= f'(\xi^*_{pj}) \sum_q w_{jq} \left[\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial \xi^*_{pq}}\right] \;= \\
&= f'(\xi^*_{pj}) \sum_q w_{jq} \sigma^*_{pq}\,,
\end{aligned} \tag{4.68}$$

For the considered 'bipolar' BP-network model and the hidden neuron j, $f'(\xi_{pj}) = (1 - y^2_{pj})$ and $f'(\xi^*_{pj}) = (1 - y^{*2}_{pj})$.

Altogether:

$$\frac{\partial G^1_p}{\partial w_{ij}} \;=\; \sigma_{pj} y_{pi} - \sigma^*_{pj} y^*_{pi}\,, \tag{4.69}$$

where the terms σ_{pj} and σ^*_{pj} are computed recursively according to:

$$\sigma_{pj} = \begin{cases} \left(\sum_u \frac{1}{(x_{pu} - x^*_{pu})^2}\right)(y_{pj} - y^*_{pj})\,, & j \in L_{l+2} \\ (1 - y^2_{pj}) \sum_q w_{jq} \sigma_{pq}\,, & j \in L_k\,,\; 1 < k \le (l+1)\,, \end{cases} \tag{4.70}$$

$$\sigma^*_{pj} = \begin{cases} \left(\sum_u \frac{1}{(x_{pu} - x^*_{pu})^2}\right)(y_{pj} - y^*_{pj})\,, & j \in L_{l+2} \\ (1 - y^{*2}_{pj}) \sum_q w_{jq} \sigma^*_{pq}\,, & j \in L_k\,,\; 1 < k \le (l+1)\,. \end{cases} \tag{4.71}$$

In the above equations, q indexes neurons in the layer above the neuron j.

An advantage of the above-defined rules for weight adjustment corresponding to $G^1(\vec{w})$ is, that they are relatively simple and may not much slower down the training process. Although there are two recursive terms σ_{pj} and σ^*_{pj}, their complexity is similar to the complexity of the recursive terms δ_{pj} corresponding the performance error function (defined by Equation (1.18)).

Problems of the SCA1-method A serious problem of the sensitivity error function $G^1(\vec{w})$ is, that it requires all denominators Δx_{pu} to be different from zero. Otherwise, the expressions (4.53), (4.54) and (4.69) can't be evaluated. However, even if $\Delta x_{pu} \neq 0$ for all training patterns p and network inputs u, the computation of the error function $G^1(\vec{w}) = 1/2 \sum_{p,u,v} (T^p_{uv})^2$ and the corresponding rules for weight adjustment can lead to serious rounding problems. The reason is, that the approximative sensitivity measure $T^p_{uv} = \Delta y_{pv} / \Delta x_{pu}$ computes the division of each couple of input and output dimensions separately.

When computing $\frac{\partial G^1_p}{\partial w_{ij}}$, the worse rounding problems occur in the evaluation of the sub-terms $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$. If for a certain training pattern p and a network input u, the term Δx_{pu} is very close to zero, the terms $\frac{1}{\Delta^2 x_{pu}}$ will yield values of a high order of magnitude. If x^*_{pu} is created by adding random noise within the range of 0.01-1% to x_{pu}, the value of $\frac{1}{\Delta^2 x_{pu}}$ is from the range $10^4 - 10^8$. Also the terms $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$ thus have very different orders of magnitude for the respective training patterns, typically from the range $10^5 - 10^8$.

We will illustrate the issue on an example of a training set with 192 training patterns and 18 input features (this training set corresponds to the **BIN2** data set that is described in Subsection 5.1.1 on page 114). Table 4.1 contains the average numbers of training patterns with the value of $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$ with the given order of magnitude.

Table 4.1: Average numbers of training patterns, for which the value of $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$ has the given order of magnitude.

order of magnitude	10^5	10^6	10^7	10^8	all
average number of training patterns	7.1	91.8	91.2	2.0	192

As a result, the adaptation rules for $G^1(\vec{w})$ extremely overrate training patterns, for which Δx_{pu} is close to zero for some of the inputs and thus $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$ has a higher-than-average order of magnitude (e.g., 10^8 in the above example). For such training patterns, the value of $\left|\frac{\partial G^1_p}{\partial w_{ij}}\right|$ is at least one or two orders of magnitude higher than for an average training pattern. Therefore, a small number of such extreme training patterns entirely defuse the impact of all other training patterns on weight adaptation during training.

Example: Although the rounding problem of the SCA-method becomes more serious with the growing number of input features, it can be illustrated also on the following example of a BP-network with just two inputs, one output and an arbitrary internal structure. In such a case, the approximative sensitivity error function corresponding to a given training pattern p has the following form:

$$2\, G^1_p(\vec{w}) = \left(\frac{\Delta y_{p1}}{\Delta x_{p1}}\right)^2 + \left(\frac{\Delta y_{p1}}{\Delta x_{p2}}\right)^2 = \frac{(y_{p1} - y^*_{p1})^2}{(x_{p1} - x^*_{p1})^2} + \frac{(y_{p1} - y^*_{p1})^2}{(x_{p2} - x^*_{p2})^2}\,. \qquad (4.72)$$

Figure 4.2 and Table 4.2 illustrate, how the terms Δx_{p1}, Δx_{p2} and Δy_{p1} can look like for concrete training patterns. Table 4.2 contains also the corresponding

values of the error function $G_p^1(\vec{w})$ and its subterms.

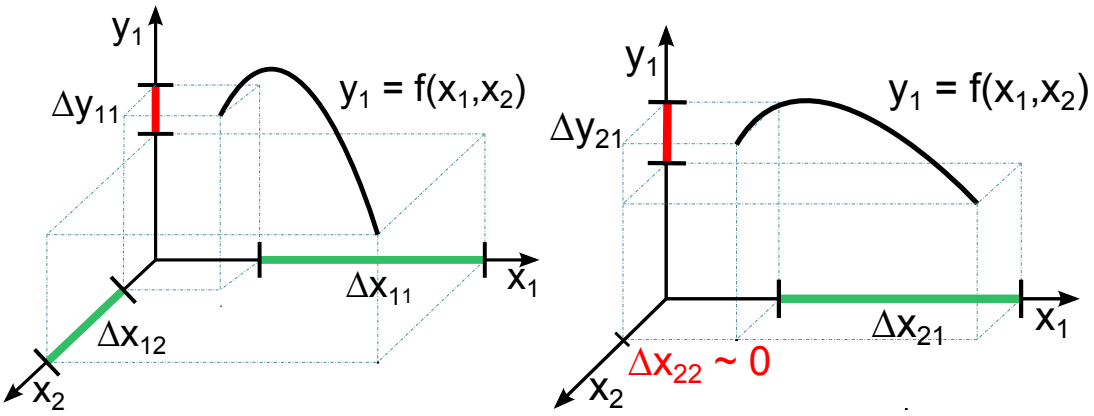


Figure 4.2: The graphs of the network function f and of the terms Δx_{p1}, Δx_{p2} and Δy_{p1} for two considered training patterns ($p = 1, 2$). The terms Δx_{p1}, Δx_{p2} are depicted by green lines and Δy_{p1} are depicted by red lines.

The graph on the left side of Figure 4.2 ($p = 1$) corresponds to a training pattern, where the terms Δx_{11}, Δx_{12} and Δy_{11} have the same orders of magnitude. The graph on the right ($p = 2$) corresponds to a training pattern, where $\Delta x_{22} \sim 0$ (it has smaller order of magnitude than Δx_{21}). In the second case, the absolute values of $G_2^1(\vec{w})$ and $\frac{\partial G_2^1}{\partial w_{ij}}$ will be extremely high when compared to the first example (regardless of the concrete values of Δx_{21} and Δy_{21}). Therefore, the second training pattern will have an unreasonably great impact on weight adaptation when compared to the first one.

Table 4.2: Values of the error function $G_p^1 = \frac{1}{2}\left(\frac{\Delta^2 y_{p1}}{\Delta^2 x_{p1}} + \frac{\Delta^2 y_{p1}}{\Delta^2 x_{p2}}\right)$ and its subterms for three representative training patterns (indexed by p) for a simple BP-network with no hidden neurons and a single output neuron with the linear transfer function and the weights $w_{01} = 0.1$, $w_{11} = 0.9$ and $w_{21} = -0.1$.

p	x_{p1}	x_{p2}	y_{p1}	Δx_{p1}	Δx_{p2}	Δy_{p1}	$\frac{1}{\Delta^2 x_{p1}}$	$\frac{1}{\Delta^2 x_{p2}}$	$\Delta^2 y_{p1}$	$\frac{1}{\Delta^2 x_{p1}} + \frac{1}{\Delta^2 x_{p2}}$	G_p^1
1	0	1	0	$-1.66 \cdot 10^{-3}$	$9.09 \cdot 10^{-3}$	$2.40 \cdot 10^{-3}$	$3.64 \cdot 10^5$	$1.21 \cdot 10^4$	$5.76 \cdot 10^{-6}$	$3.76 \cdot 10^5$	1.1
2	-1	1	-0.9	$4.11 \cdot 10^{-3}$	$-7.43 \cdot 10^{-3}$	$-4.44 \cdot 10^{-3}$	$5.93 \cdot 10^4$	$1.81 \cdot 10^4$	$1.97 \cdot 10^{-6}$	$7.74 \cdot 10^4$	0.8
3	1	1	0.9	$-6.68 \cdot 10^{-3}$	$3.34 \cdot 10^{-4}$	$6.05 \cdot 10^{-3}$	$2.24 \cdot 10^4$	$8.95 \cdot 10^6$	$3.66 \cdot 10^{-5}$	$8.97 \cdot 10^6$	164.1

The same problem is shown on the third row of Table 4.3. Because for the third training pattern, the values of Δx_{p1} and Δx_{p2} have different orders of magnitude, its impact on weight adjustment is overrated (as indicated by the value of $G_p^1(\vec{w})$).

The final version of the SCA-method

To avoid serious rounding problems during training of the SCA1 method, we decided to replace the error function $G^1(\vec{w}) = \frac{1}{2}\sum_{p,u,v}\left(\frac{\Delta y_{pv}}{\Delta x_{pu}}\right)^2$ (defined by Equation (4.54)) by an alternative, more robust expression. In the following paragraph, we will discuss the new form of the approximative sensitivity error function.

To avoid the rounding problem, the error function and the terms for weight adjustment shouldn't include the sub-terms $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$. To achieve this requirement, we won't process each pair of input and output dimensions separately as it is done by the approximative sensitivity measure $T^p_{uv} = \Delta y_{pv}/\Delta x_{pu}$ (Equation (4.53)). Instead, we will project all the network inputs into a single dimension using the squared (Euclidean) distance. We will do the same also with network outputs. In such a case, a zero or small value of Δx_{pu} for a single dimension of the input won't cause problems.

The new approximative sensitivity error function G^A [81] thus has the form

$$G^A(\vec{w}) \;=\; \frac{1}{2}\sum_p \frac{\sum_v \Delta^2 y_{pv}}{\sum_u \Delta^2 x_{pu}} \;=\; \sum_p G^A_p(\vec{w})\,, \tag{4.73}$$

where $G^A_p(\vec{w}) \;=\; \frac{1}{2}\frac{\sum_v \Delta^2 y_{pv}}{\sum_u \Delta^2 x_{pu}}$ is the approximative sensitivity error function corresponding to the training pattern p. G^A expresses for the training patterns the ratio between the squared (Euclidean) distances of the outputs, $\sum_v \Delta^2 y_{pv}$, and the squared (Euclidean) distances of the inputs, $\sum_u \Delta^2 x_{pu}$.

Example: We will illustrate the form of the approximative sensitivity error function G^A on the example of a BP-network with just two inputs, one output and an arbitrary internal structure. In such a case, the approximative sensitivity error function corresponding to a given training pattern p has the following form:

$$2\,G^A_p(\vec{w}) = \frac{\Delta^2 y_{p1}}{\Delta^2 x_{p1} \;+\; \Delta^2 x_{p2}} \;=\; \frac{(\,y_{p1} - y^*_{p1}\,)^2}{(\,x_{p1} \;-\; x^*_{p1}\,)^2 \;+\; (\,x_{p2} \;-\; x^*_{p2}\,)^2}\,. \tag{4.74}$$

Figure 4.3 illustrates, how the terms Δy_{p1} and $\sqrt{\Delta^2 x_{p1} \;+\; \Delta^2 x_{p2}}$ can look like for a concrete training pattern.

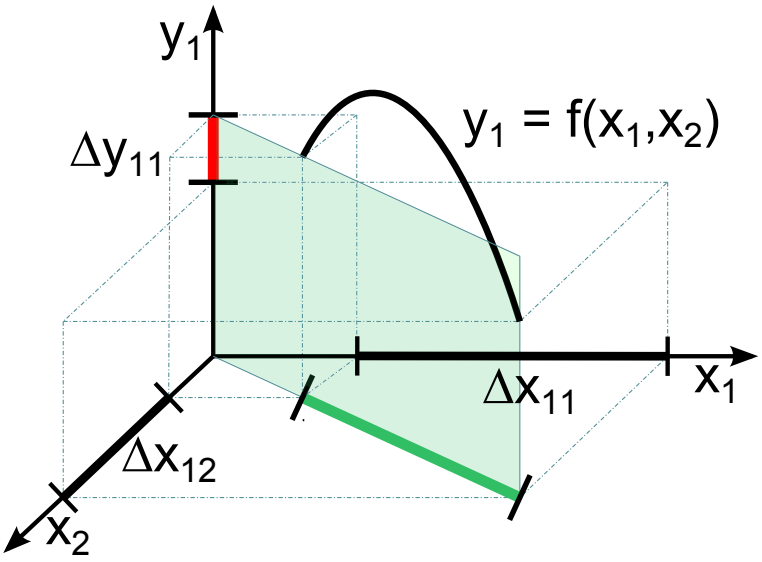


Figure 4.3: The graph of the network function f and of the terms Δx_{p1}, Δx_{p2} (depicted by black lines), $\sqrt{\Delta^2 x_{p1} + \Delta^2 x_{p2}}$ (depicted by green line) and Δy_{p1} (depicted by red line) for a certain training pattern ($p = 1$).

Table 4.3: Values of the error functions $G_p^1 = \frac{1}{2}\left(\frac{\Delta^2 y_{p1}}{\Delta^2 x_{p1}} + \frac{\Delta^2 y_{p1}}{\Delta^2 x_{p2}}\right)$ and $G_p^A = \frac{1}{2}\frac{\Delta^2 y_{p1}}{\Delta^2 x_{p1} + \Delta^2 x_{p2}}$ for three representative training patterns (indexed by p) and a simple BP-network with no hidden neurons and a single output neuron with the linear transfer function and the weights $w_{01} = 0.1$, $w_{11} = 0.9$ and $w_{21} = -0.1$.

p	x_{p1}	x_{p2}	y_{p1}	Δx_{p1}	Δx_{p2}	Δy_{p1}	$\frac{1}{\Delta^2 x_{p1}} + \frac{1}{\Delta^2 x_{p2}}$	$\frac{1}{\Delta^2 x_{p1} + \Delta^2 x_{p2}}$	$\Delta^2 y_{p1}$	G_p^1	G_p^A
1	0	1	0	$-1.66 \cdot 10^{-3}$	$9.09 \cdot 10^{-3}$	$2.40 \cdot 10^{-3}$	$3.76 \cdot 10^{5}$	$1.17 \cdot 10^{4}$	$5.76 \cdot 10^{-6}$	1.1	0.03
2	−1	1	−0.9	$4.11 \cdot 10^{-3}$	$-7.43 \cdot 10^{-3}$	$-4.44 \cdot 10^{-3}$	$7.74 \cdot 10^{4}$	$1.39 \cdot 10^{4}$	$1.97 \cdot 10^{-6}$	0.8	0.14
3	1	1	0.9	$-6.68 \cdot 10^{-3}$	$3.34 \cdot 10^{-4}$	$6.05 \cdot 10^{-3}$	$8.97 \cdot 10^{6}$	$2.23 \cdot 10^{4}$	$3.66 \cdot 10^{-5}$	164.1	0.41

Table 4.3 shows a comparison of the values of the error functions $G_p^A(\vec{w})$ and $G_p^1(\vec{w})$ (defined by Equation (4.72)) for three representative training patterns (indexed by p). As shown on the third row of Table 4.3, even if $\Delta x_2 \sim 0$ (i.e., if Δx_1 and Δx_2 had different orders of magnitude) for a given training pattern, it wouldn't cause rounding problems when computing $G_p^A(\vec{w})$. Moreover, the impact of such training pattern on weight adjustment wouldn't be overrated.

The overall error function $H(\vec{w}) = E(\vec{w}) + c_F F(\vec{w}) + c_G G^A(\vec{w})$ will be evaluated according to:

$$H(\vec{w}) = \frac{1}{2}\sum_p \sum_v (\, y_{pv} - d_{pv}\,)^2 \;+\; c_F \sum_p \sum_{j'} (1 + y_{pj'})^s \,(1 - y_{pj'})^s \, y_{pj'}^2 \;+ \\ + c_G \, \frac{1}{2} \sum_p \frac{\sum_v \Delta^2 y_{pv}}{\sum_u \Delta^2 x_{pu}}\,, \tag{4.75}$$

where j' indexes all hidden neurons, d denotes the desired activity of a neuron and s is a parameter for tuning the shape of the representation error function. The coefficients c_F and c_G reflect the trade-off between the influence of E, F and G^A in H.

The derivative of the overall error function $H'(\vec{w}) = E'(\vec{w}) + c_F F'(\vec{w}) + c_G G^{A'}(\vec{w})$ is computed as: $H'(\vec{w}) = \sum_p H'_p(\vec{w})$, where $H'_p(\vec{w}) = (..., \frac{\partial H_p}{\partial w_{ij}}, ...)$ and

$$\frac{\partial H_p}{\partial w_{ij}} \;=\; \frac{\partial E_p}{\partial w_{ij}} + c_F \frac{\partial F_p}{\partial w_{ij}} + c_G \frac{\partial G_p^A}{\partial w_{ij}}\,. \tag{4.76}$$

The partial derivatives in $\partial E_p/\partial w_{ij} + c_F \partial F_p/\partial w_{ij}$ were already stated in Equation (4.15). It remains to determine the terms $\partial G_p^A/\partial w_{ij}$.

The derivation of the rules for weight adjustment The derivation of the rules for weight adjustment with respect to the error function G^A (defined by (4.73) is analogical to the SCA1-method and the error function G^1 (see Subsection 4.3.2 and equations (4.57), ...,(4.68)).

Let $(\vec{x}_p, \vec{d}_p)$ be the training pattern presented to the BP-network in time t and $\vec{x}_p^*$ be the noise-corrupted input pattern. Let i be the neuron connected with neuron j via the weight w_{ij}.

In the following paragraphs, v will index all output neurons and u will index all input neurons. All neurons from the layer above the neuron j will be declared as

those indexed by q. The activity and the potential of a neuron j corresponding to the input pattern $\vec{x}_p$ will be denoted as y_{pj} and ξ_{pj}, respectively. The activity and the potential of a neuron j corresponding to the noise-corrupted input pattern $\vec{x}^*_p$ will be denoted as y^*_{pj} and ξ^*_{pj}, respectively. For the output neurons v, $\Delta y_{pv} = y_{pv} - y^*_{pv}$. For the input neurons u, their activity is equal to their input (i.e., $y_{pu} = x_{pu}$, $y^*_{pu} = x^*_{pu}$), $\Delta x_{pu} = x_{pu} - x^*_{pu}$.

The terms for $\frac{\partial G^A_p}{\partial w_{ij}}$ can be computed using the sum and chain rules:

$$\frac{\partial G^A_p}{\partial w_{ij}} \;=\; \frac{\partial}{\partial w_{ij}} \left(\frac{1}{2} \frac{\sum_v \Delta^2 y_{pv}}{\sum_u \Delta^2 x_{pu}} \right) \;=\; \frac{1}{2} \left(\frac{1}{\sum_u \Delta^2 x_{pu}} \right) \sum_v \frac{\partial (\Delta^2 y_{pv})}{\partial w_{ij}} \tag{4.77}$$

Because

$$\begin{aligned} \frac{\partial (\Delta^2 y_{pv})}{\partial w_{ij}} &= 2\, \Delta y_{pv} \frac{\partial (\Delta y_{pv})}{\partial w_{ij}} \;=\; 2\, \Delta y_{pv} \left(\frac{\partial y_{pv}}{\partial w_{ij}} \;-\; \frac{\partial y^*_{pv}}{\partial w_{ij}} \right) \;= \\ &= 2\, \Delta y_{pv} \left(\frac{\partial y_{pv}}{\partial \xi_{pj}} \frac{\partial \xi_{pj}}{\partial w_{ij}} \;-\; \frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} \frac{\partial \xi^*_{pj}}{\partial w_{ij}} \right) , \end{aligned} \tag{4.78}$$

we obtain:

$$\begin{aligned} \frac{\partial G^A_p}{\partial w_{ij}} = & \left(\frac{1}{\sum_u \Delta^2 x_{pu}} \right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial \xi_{pj}} \frac{\partial \xi_{pj}}{\partial w_{ij}} \;- \\ & - \left(\frac{1}{\sum_u \Delta^2 x_{pu}} \right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} \frac{\partial \xi^*_{pj}}{\partial w_{ij}} \,. \end{aligned} \tag{4.79}$$

The auxiliary terms σ_{pj} and σ^*_{pj} for the back-propagated sensitivity errors will be defined as:

$$\begin{aligned} \sigma_{pj} &= \left(\frac{1}{\sum_u \Delta^2 x_{pu}} \right) \sum_v \Delta y_{pv} \frac{\partial y_{pv}}{\partial \xi_{pj}} \,, \\ \sigma^*_{pj} &= \left(\frac{1}{\sum_u \Delta^2 x_{pu}} \right) \sum_v \Delta y_{pv} \frac{\partial y^*_{pv}}{\partial \xi^*_{pj}} \end{aligned} \tag{4.80}$$

Because $\frac{\partial \xi_{pj}}{\partial w_{ij}} = y_{pi}$ and $\frac{\partial \xi^*_{pj}}{\partial w_{ij}} = y^*_{pi}$, we get:

$$\frac{\partial G^A_p}{\partial w_{ij}} \;=\; \sigma_{pj} y_{pi} - \sigma^*_{pj} y^*_{pi} \,, \tag{4.81}$$

where the terms σ_{pj} and σ^*_{pj} can be computed recursively according to:

$$\sigma_{pj} = \begin{cases} \left(\frac{1}{\sum_u (x_{pu} - x^*_{pu})^2} \right) (y_{pj} - y^*_{pj}) \,, & j \in L_{l+2} \\ (1 - y^2_{pj}) \sum_q w_{jq} \sigma_{pq} \,, & j \in L_k \,,\; 1 < k \leq (l+1) \,, \end{cases} \tag{4.82}$$

$$\sigma^*_{pj} = \begin{cases} \left(\frac{1}{\sum_u (x_{pu} - x^*_{pu})^2} \right) (y_{pj} - y^*_{pj}) \,, & j \in L_{l+2} \\ (1 - y^{*2}_{pj}) \sum_q w_{jq} \sigma^*_{pq} \,, & j \in L_k \,,\; 1 < k \leq (l+1) \,. \end{cases} \tag{4.83}$$

In the above equations, q indexes all neurons in the layer above the neuron j.

The rules for weight adjustment corresponding to $G^A(\vec{w})$ differ from the rules for $G^1(\vec{w})$ (defined by Equations (4.69), (4.70) and (4.71)) just in small details. The derivation of the terms for σ_{pj} and σ^*_{pj} (Equations (4.82) and (4.83)) is analogical to the SCA1-method (Subsection 4.3.2). We just substitute all the sub-terms $\left(\sum_u \frac{1}{\Delta^2 x_{pu}}\right)$ by the sub-terms $\left(\frac{1}{\sum_u \Delta^2 x_{pu}}\right)$.

Although there are two recursive error terms σ_{pj} and σ^*_{pj}, the overall complexity of the adaptation rules for $G^A(\vec{w})$ is similar to the complexity of the adaptation rules for the performance error function (defined by Equations (1.17) and (1.18)). For this reason, the SCA-regularization technique is expected to be very efficient when compared to the analytical SC-regularization technique and may not much slower down each iteration of the training process.

The SCGSA-training algorithm The last version of our framework, the so called SCGSA-training algorithm [81], combines the SCA- and IR-regularization techniques with the SCG-training algorithm (Algorithm 4.2 on page 80) and with our training-and-pruning methodology (Algorithms 4.3 and 4.4 on pages 89 and 90).

The training process of the SCGSA-method is identical to the standard SCG-algorithm (see Algorithm 4.2 on page 80 for the general schema of the SCG-like algorithms), however with $H(\vec{w}) = E(\vec{w}) + c_F F(\vec{w}) + c_G G^A(\vec{w})$ (defined by (4.75)) as the error function. The derivative of $H(\vec{w})$ is computed as: $H'(\vec{w}) = \sum_p H'_p(\vec{w})$, where $H'_p(\vec{w}) = (..., \frac{\partial H_p}{\partial w_{ij}}, ...)$. For the considered 'bipolar' BP-network model, the terms $\partial H_p / \partial w_{ij}$ correspond to:

$$\frac{\partial H_p}{\partial w_{ij}} = \begin{cases} (y_{pj} - d_{pj})\, y_{pi} + c_G \left(\sigma_{pj} y_{pi} - \sigma^*_{pj} y^*_{pi}\right), \\ \qquad\qquad \text{for neuron } j \in L_{l+2} \\ \left\{ \sum_q \delta_{pq}\, w_{jq} \;+\; 2\, c_F \left[1 - (s+1)\, y_{pj}^2\right] (1 - y_{pj}^2)^{s-1}\, y_{pj} \right\} \cdot \\ \qquad \cdot (1 - y_{pj}^2)\, y_{pi} \;+\; c_G \left(\sigma_{pj} y_{pi} - \sigma^*_{pj} y^*_{pi}\right), \\ \qquad\qquad \text{for neuron } j \in L_{l+1} \\ \left\{ \sum_q (\delta_{pq} + c_F\, \varrho_{pq})\, w_{jq} \;+\; 2\, c_F \left[1 - (s+1)\, y_{pj}^2\right] (1 - y_{pj}^2)^{s-1}\, y_{pj} \right\} \cdot \\ \qquad \cdot (1 - y_{pj}^2)\, y_{pi} \;+\; c_G \left(\sigma_{pj} y_{pi} - \sigma^*_{pj} y^*_{pi}\right), \\ \qquad\qquad \text{for neuron } j \in L_k\,,\ 2 \le k \le l. \end{cases} \tag{4.84}$$

In the above expressions, s denotes the parameter for tuning the shape of the representation error function, q indexes the neurons in the layer above the neuron j. c_F and c_G are constants representing the influence of the respective error terms. δ_{pq} can be determined according to (1.18), ϱ_{pq} according to (4.14), σ_{pj} according to (4.82) and σ^*_{pj} according to (4.83).

The SCGSA-method is summarized in Algorithm 4.6.

4.3.3 Summary of Section 4.3

In this section, we designed a new approximative sensitivity-based regularization technique SCA. It replaced in our framework the formerly-proposed SC-regularization technique for analytical sensitivity control. The main reasons for the

Algorithm 4.6 Function $SCGSA()$ (Scaled conjugate gradients algorithm with learning internal representation and approximative sensitivity control):

1. *Input:*
   BP-network $M = (N, C, I, O, w, t)$ with the weight vector $\vec{w}$ of the size W.
   A training set $T = \{ (\vec{x}_p, \vec{d}_p) \mid \quad \vec{x}_p \in \Re^n, \quad \vec{d}_p \in \Re^m, \quad p \in \{1, \ldots, P\}\}$.
2. *Initialize:*
   For each training input pattern $\vec{x}_p \in T$, create a new input pattern $\vec{x_p^*} \in \Re^n$ by adding a small amount of random noise (within the range of 0.01-1%) to each input feature.
3. *Train:*
   Train the BP-network M on the training set T using the SCG-training algorithm (described by Algorithm 4.2 on page 80) and the overall error function $H(\vec{w}) = \sum_p H_p(\vec{w})$ defined by (4.75) with $\frac{\partial H_p}{\partial w_{ij}}$ defined by (4.84).
   $M' = SCG(M, T, H)$.
4. *Return:* M'.

change were great computational costs of the SC-technique. The SCA-regularization technique is expected to be very fast when compared to SC-method. Despite of the approximative nature of the SCA-method, it should keep the good qualities of the SC-method as much as possible:

- It should simplify pruning based on sensitivity analysis.
- It should contribute to a smoother network function and better generalization ability of the trained BP-networks.

The new-proposed SCGSA method (Algorithm 4.6) combines SCA with the SCG-training algorithm, our formerly defined training-and-pruning methodology and the IR-regularization technique. The qualities and drawbacks of the SCGSA-method will by assessed in the experimental part of this work in Chapter 5. For a detailed experimental evaluation, see also [81].

5. Experiments

5.1 Introduction

This chapter is devoted to experimental evaluation of our framework. In our experiments, we concentrated mainly on the following issues:

1. To verify the behavior and the required abilities of our framework on various types of the data.
2. To assess the benefits and drawbacks of the introduced methods when compared with related techniques.

The experiments were performed on a 2.8 GHz quad core processor, 12 GB RAM. Our system was implemented in Matlab® 7.0.1 [72] and used a single processor and 2 GB RAM. Our implementation uses functions from the Neural Network Toolbox™5.0 [26] (especially, for the standard BP-algorithm and the SCG-training algorithm). The methods for training of BP-networks, that were not included in the Neural Network Toolbox™, were newly implemented (including the algorithms for learning from hints, learning internal representation and all the new-proposed techniques). The created library is available as an attachment of this thesis.

5.1.1 Data sets

The experiments involved several data sets of various properties – discrete and continuous, artificial and real-world, simple and complex. Especially for the real-world data, an advanced sophisticated preprocessing had to be done. To examine, whether the tested methods are able to recognize relevant input features, some of the data sets were enhanced by several irrelevant, randomly generated input features.

Generally, our experiments concern three tasks, which we denoted as *Binary Addition*, *Binary Multiplication* and *World Bank*. In the following paragraphs, we will describe these tasks and the corresponding data sets in detail.

Binary addition of two 3-bit numbers

For the *Binary Addition* task, we created two data sets: **BIN2A** and **BIN2**.

The **BIN2A** data set consists of all 64 possible training patterns with 6 bipolar input features and 4 bipolar output features. Each input pattern consists of the bipolar codes of the two numbers to be added. The bipolar code of their sum represents the desired output. When adding $4(\sim (+1,-1,-1))$ and $7(\sim (+1,+1,+1))$ yielding the sum $11(\sim (+1,-1,+1,+1))$, the corresponding training pattern would have the form: $[[+1,-1,-1,+1,+1,+1],[+1,-1,+1,+1]]$. All of the 64 patterns formed the training set. As the optional hint output, we provided the carry-information to the second output bit.

The **BIN2** data set consists of 320 patterns with 18 bipolar input features and 4 bipolar output features. The first 6 input features represent two three-bit binary numbers and the four bits of the output indicate the sum of the two binary numbers. The other 12 input features are bipolar bits generated randomly with a uniform distribution. When adding $4(\sim (+1,-1,-1))$ and $7(\sim (+1,+1,+1))$ yielding the sum $11(\sim (+1,-1,+1,+1))$, the corresponding training pattern would have the form: $[[+1,-1,-1,+1,+1,+1,...],[+1,-1,+1,+1]]$.

The **BIN2** data set is divided into the training set of the size 192, the validation set and the test set, both of the size 64. Each of the 64 possible training patterns is present three-times in the training set and once in the other two subsets. The fact, that 2/3 of the input features are irrelevant (randomly generated), makes the **BIN2** task much more difficult than the **BIN2A** task.

Binary multiplication of two 3-bit numbers

The first data set for the *Binary Multiplication* task is denoted as **BIN3**. The **BIN3** data set consists of 320 patterns with 18 bipolar input features and 6 bipolar output features. 192 of the patterns are contained in the training set, 64 in the validation set and 64 in the test set.

The first 6 input features stand for the two three-bit binary numbers to be multiplied and the six bits of the output indicate the product of the two factors. The other 12 input features are bipolar bits generated randomly with a uniform distribution. When multiplying, e.g., $4(\sim (+1,-1,-1))$ and $7(\sim (+1,+1,+1))$ yielding $28(\sim (-1,+1,+1,+1,-1,-1))$, the corresponding training pattern would have the form: $[[+1,-1,-1,+1,+1,+1,...],[-1,+1,+1,+1,-1,-1]]$.

All of the 64 possible training patterns are present 3-times in the training set and once in the other two subsets. *Binary Multiplication* is indeed a rather difficult task due to the unbalanced number of -1 and 1 at the outputs (1365 : 555). In addition, 2/3 of the input features of the **BIN3** data set are irrelevant (randomly generated).

Binary multiplication of two 2-bit numbers

The second data set for the *Binary Multiplication* task is denoted as **BIN3A**. It consists of 16 patterns with 4 bipolar input features and 4 bipolar output features. All of the 16 patterns form the training set. The 4 input features are the two two-bit binary numbers to be multiplied and the four bits of the output indicate the product of the two factors. When multiplying, e.g., $2(\sim (+1,-1))$ and $3(\sim (+1,+1))$ yielding $6(\sim (-1,+1,+1,+1))$, the corresponding training pattern would have the form: $[[+1,-1,+1,+1],[-1,+1,+1,+1]]$.

World Bank

The *World Bank* task is based on continuous real-world data obtained from the World Bank [109]. The collected data set consists of 972 patterns comprising 25 World development indicators (WDI-indicators) for each of the respective 162 countries and over the years 2001-2006. Each pattern has thus 25 input features listed in Table 5.1. There are two alternative output features: PPP-GNI (with continuous numerical values) and the so-called Income group (with 5 possible discrete values).

For the *World Bank* task, we created two data sets: **WBA** (a regression task) and **WB** (a classification task):

- The **WBA** data set consists of 956 patterns with 25 numerical input features encoding the WDI-indicators (listed in Table 5.1). As the output feature, we have used the PPP-GNI indicator.

Table 5.1: World development indicators

	Input features:
1	Purchasing power parity conversion factor (local currency units per international dollar)
2	Ratio of purchasing power parity conversion factor to official exchange rate
3	Present value of debt(% of exports of goods, services nd income)
4	Short-term debt (% of total debt)
5	Short-term debt (% of exports of goods, services, and income)
6	Total debt service (% of gross national income)
7	GINI-Index
8	Tax revenue collected by central government (% of gross domestic product)
9	Taxes on income, profits, and capital gains (% of revenue)
10	Taxes on goods and services (% of revenue)
11	Taxes on international trade (% of revenue)
12	Social contributions (% of revenue)
13	Expenditures for Research & Education (% of gross domestic product)
14	GDP deflator (implicit price deflator for gross domestic product, % growth)
15	Fertility rate, total (births per woman)
16	Fixed line and mobile phone subscribers (per 1,000 people)
17	GDP growth (annual percentage growth rate of gross domestic product)
18	High-technology exports (% of manufactured exports)
19	Inflation (as measured by the GDP deflator, annual %)
20	Internet users (per 1,000 people)
21	Life expectancy at birth, total (years)
22	Military expenditure (% of gross domestic product)
23	Poverty head-count ratio at national poverty line (% of population)
24	Present value of debt (% of gross national income)
25	Total debt service (% of exports of goods, services and income)
	Output features:
26	PPP-GNI (Gross national income converted to international dollars using purchasing power parity rates)
27	Income group (High income OECD, High income nonOECD, Upper middle income, Lower middle income, Low income)

In the experiments with hints, we used the Income group (*hintIG*) to create 5 hint outputs that label the five classes of the Income Groups. The other option was to use clustering of the input data into k groups as a hint. For clustering, the c-means algorithm with Euclidean distance measure has been applied.

- The **WB** data set consists of 956 patterns with 35 numerical input features. 25 of the input features encode the WDI-indicators (listed in Table 5.1). The other 10 input features were generated randomly (each from the interval $[-1, 1]$) with a uniform distribution. There are 5 output features that label the five classes of the Income Groups.

Table 5.2 briefly summarizes the basic characteristics of the respective data sets.

Table 5.2: Data sets and their characteristics.

data set	number of				type of data		
	training patterns	output features	all input features	random input features	artificial/ real	discrete/ continuous	simple/ medium/ complex
BIN2A	64	4	6	–	artificial	discrete	simple
BIN2	320	4	18	12	artificial	discrete	medium
BIN3	320	6	18	12	artificial	discrete	complex
BIN3A	16	4	4	–	artificial	discrete	simple
WBA	956	1	25	–	real	continuous	complex
WB	956	5	35	10	real	continuous	complex

Preprocessing of the *World Bank* data

The data sets corresponding to the *World Bank* task required an advanced preprocessing. In the following paragraphs, we will describe the whole process in detail.

At first, we had to decide, what would be the input and output features, and acquire the data from the World Bank. As a task, we decided to predict the gross national income of the particular countries over the years 2001-2006. It is quantified by the indicators PPP-GNI and Income group. Then, we had to select an informative set of input features that would facilitate the prediction. The World Bank publishes annually hundreds of WDI-indicators. From this amount, we selected 25 WDI-indicators to be the input features, while we followed our knowledge of the domain.

The acquirement of the data from the World Bank was a challenging task, because the values of the chosen WDI-indicators were not simply available in one source. On the contrary, the data was spread over a high number of tables and text documents with various formatting. For this reason, we collected the data manually and converted it into a single text format that can be processed by Matlab®.

A further problem of the original data was, that some of the WDI-Indicators were available just as total values that can hardly be mutually compared (e.g., the total debt service of a country, the total number of Internet users in a country).

Therefore, we transformed the values of such WDI-indicators in a way to relate them to the appropriate basic entities (e.g., as a % of gross national income, as a % of revenue, or per 1,000 people).

Another serious problem of the original data was its sparsity. To solve this problem, we firstly removed from the data patterns without output features and patterns with less than 50% of input features. There remained 956 patterns from the original 1350 ones. After that, we estimated the remaining missing values either from known values (using linear interpolation) or we replaced them by mean value.

The data was normalized such that in every dimension the mean was 0 and standard deviation was 1. We used the following formula:

$$x_{pi}(new) = \frac{x_{pi} - \text{mean}_j\, x_{pj}}{\text{std}_j\, x_{pj}}, \tag{5.1}$$

where i and j index the n input features, p indexes the training patterns, x is the original value of a an input feature and $x(new)$ is its new value. The standard deviation (std) is defined by:

$$\underset{i}{\text{std}}\, x_{pi} = \left[\frac{1}{n-1} \sum_{i=1}^{n} \left(x_{pi} - \underset{j}{\text{mean}}\, x_{pj} \right)^2 \right]^{\frac{1}{2}}. \tag{5.2}$$

The same normalization was done also for the output feature *PPP-GNI*. For the *Income Group* indicator, we created 5 output features that label the five classes of the Income groups (with the values 1 and -1).

5.1.2 Performance evaluation

To compare the performance of the tested methods, several approaches were used depending on the character of the respective data sets. Typically, the data sets were divided into the training, validation and test sets. The training set was used for training. The validation set was used by the early stopping strategy (for a detailed description, see Section 3.4.1) and by the training-and-pruning methodology (Algorithms 4.3 and 4.4). The test set was used to estimate the generalization ability of the trained BP-networks (measured, e.g., as the mean squared error (MSE, defined by Equation 1.9) on the test set).

The **BIN2** and **BIN3** data sets were divided into the training, validation and test sets a priori. The **BIN2A** data set was not divided – all of the training patterns formed the training set. During experimental testing, each tested training algorithm was evaluated on the same set of 100 different randomly initialized networks (with the weights from the interval $[-1, 1]$). For each of the 100 trained BP-networks, we computed the values of the tested parameters (e.g., of the mean squared error on the test set or of the number of epochs) and averaged them over the 100 experiments.

For the **WBA** and **WB** data sets, we kept $\frac{1}{10}$ of the original data as a validation set and used the remaining $\frac{9}{10}$ for training and testing. To compare the performance of tested methods, we used the 10-fold cross-validation [74] (alternatively, the 10-times repeated 10-fold cross-validation). Similarly to the previous

case, all the tested training algorithms and their parameter settings were applied to the same set of 10 (respectively, 100) different randomly initialized BP-networks.

The principle of the k-fold cross-validation is to divide the data set into k disjoint parts with (almost) the same number of training patterns. One of the k parts forms the test set, while the other $(k-1)$ parts form the training set. There are k possible ways, how to split the data into the training and test sets. Thus, each of the tested training algorithms is repeated k-times, each time using a randomly initialized BP-network and a different pair of training and test sets. For each of the k trained BP-networks, we computed the values of the tested parameters and averaged them over the k experiments. The principle of k-fold cross-validation is summarized by Algorithm 5.1 on page 119 .

Algorithm 5.1 General principle of the k-fold cross-validation

1. *Input:*
   A training algorithm ALG.
   A data set T.
   k randomly initialized BP-networks $M_1, \ldots, M_k$.
   The tested criterion $E = E(M, T)$.
2. Remove a part (e.g., $\frac{1}{10}$) of the patterns from T to form a validation set T_v.
3. Divide T into k disjoint subsets $T_1, \ldots, T_k$ of (almost) equal sizes, where $|T_i| \geq 30$, $i = 1, \ldots, k$.
4. For $i = 1, \ldots, k$:
   (a) Form the training set T_{tr} and the test set T_t:
   $T_t = T_i$, $T_{tr} = T \setminus T_i$.
   (b) Train the BP-network M_i on the training set T_{tr} using the training algorithm ALG and the validation set T_v.
   (c) Compute the value of the tested criterion of the trained BP-network M_i on the test set T_t:
   $e_i = E(M_i, T)$.
5. Compute and return the mean and standard deviation of the values of the tested criterion over the k experiments:
   $[\mathrm{mean}_i\, e_i \pm \mathrm{std}_i\, e_i]$

5.1.3 Settings and notation

BP-network model

In our experiments, we use the so called 'bipolar' BP-network model, where all the hidden neurons have the hyperbolic tangent transfer functions, while all the output neurons implement the linear transfer functions. The weights are randomly initialized from the interval $[-1, 1]$.

Training algorithms

Table 5.3 contains the notation of the training algorithms used in the experiments. For all of the training algorithms, we applied the following stop criteria:

1. The training is stopped, if the maximal number of training epochs (indicated by the parameter *maxEpochs*) is reached.
2. The training is stopped, if the overall error on the validation set grows five times in a row. This stop criterion was omitted for the **BIN2A** data set.

When training together with pruning, we used our training-and-pruning methodology described by Algorithms 4.3 and 4.4 on pages 89 and 90. If not said otherwise, both pruning strategies based on internal representation and sensitivity analysis were applied.

When training together with learning from hints, we implemented the extra output hint method [106] (described in Subsection 3.4.2). After the BP-network was trained, we removed all the hint outputs and retrained the BP-network again.

Table 5.3: Training algorithms.

Name	Reference	Description
Reference techniques		
GD	Algorithm 1.1, Page 19	Standard back-propagation (BP-) algorithm.
GDM	Subsection 1.5.2, Page 23	BP-algorithm with momentum.
GDIR	Subsection 3.5.1, Page 67	BP-algorithm with learning internal representation.
GDMIR	Subsection 3.5.1, Page 67	BP-algorithm with momentum and with learning internal representation.
SCG	Algorithm 3.2, Page 44	Scaled conjugate gradients algorithm.
New-proposed techniques		
SCGIR	Algorithm 4.1, Page 79	Scaled conjugate gradients algorithm with learning internal representation.
SCGS	Algorithm 4.5, Page 98	Scaled conjugate gradients algorithm with learning internal representation and analytical sensitivity control.
SCGSA	Algorithm 4.6, Page 111	Scaled conjugate gradients algorithm with learning internal representation and approximative sensitivity control.
Enhancements:		
ALGIR	Subsection 4.1.2, Page 75	Algorithm ALG together with learning internal representation (with the altered IR-regularization technique).
ALGWD	Subsection 3.3.6, Page 61	Algorithm ALG together with the weight decay regularization technique.
ALG-hint	Subsection 3.4.2, Page 64	Algorithm ALG together with learning from hints.
ALG*	Subsection 3.4.3, Page 65	Algorithm ALG combined with training with jitter [95].

Table 5.4: Notation of the tested criteria.

	the average mean squared error:
MSE_{tr}	... on the training set
MSE_v	... on the validation set
MSE_t	... on the test set
$MSE(n_t)$	... on the noisy test set
	the average value of an alternative error function (e.g., the classification error):
E_{tr}	... on the training set
E_v	... on the validation set
E_t	... on the test set
$E(n_t)$	... on the noisy test set
imp	the improvement of the error when compared to the SCG method
c	the number of networks with no error on the test set
c_n	the number of networks with no error on the noisy test set
$arch$	the average number of input and hidden neurons after training (delimited by $-$)
H	the average number of hidden neurons after training
I	the average number of input neurons after training
	the number of networks that achieved during training the optimum number:
n_H	... of hidden neurons
n_A	... of both hidden and input neurons
n_I	the number of networks that pruned during training all of the input neurons corresponding to the randomly generated input features
S_t	the average (analytical) sensitivity of the trained BP-network on the test data
w_m	the average absolute value of all weights and thresholds in the BP-network
	the percentages of the activities of hidden neurons that differ from the values -1, 0, and 1 ...
p_{IR}	... at most by 0.1
$p_{IR,3}$	... at most by 0.3
	the number of networks with a well-formed condensed internal representation ...
c_R	... and no errors on the training, validation and test sets
c_A	... and an optimum number of hidden and input neurons, no errors on the training, validation and test sets
epochs	the average number of training epochs
t(s)	the elapsed training time (in seconds)

Tested criteria

Table 5.4 shows the basic notation used in the experiments to evaluate the performance of the methods. The criterion MSE corresponds to the mean squared error (defined by Equation (1.9)) averaged over all network outputs and training patterns. For the **BIN2**, **BIN2A** and **BIN3** data sets, E_{tr}, E_v, E_t and $E(n_t)$ denote the average numbers of patterns with incorrect outputs on the training, validation, test and noisy test sets, respectively. For the **WB** and **WBA** data sets, E_{tr}, E_v, E_t and $E(n_t)$ denote the classification error on the training, validation, test and noisy test sets. The noisy test set is created from the original test set by adding a random normally distributed noise (up to 5%) to all the input features.

The average (analytical) sensitivity S_t on the test data is defined as $S_t = \text{mean}_{i,v,p} |S_{iv}^p|$, where p indexes all training patterns, v indexes all output neurons, and i indexes all input and hidden neurons, the sensitivity coefficients S_{iv}^p are defined by Equations (4.16) and (4.17).

5.1.4 The structure of supporting experiments

In Chapter 4, we designed three successive versions of our general framework for training of BP-networks – the SCGIR, SCGS and SCGSA training algorithms. In the following sections, we will concentrate on an experimental evaluation, whether and how these methods fulfill the requirements that we declared in Chapter 2). We will structure this chapter based on the following fields of interest – *Generalization*, *Speed*, *Transparency* and *Structure optimization*.

5.2 Generalization

In this section, we are interested in answering of the following questions:

1. What are the prediction and generalization abilities of the trained BP-networks? Do the new-proposed methods (SCGIR, SCGS and SCGSA) favor smoother BP-network functions and do they contribute to an improved generalization ability of the trained BP-networks?
2. Are the methods sensitive to the noise in the data?

The SCGIR, SCGS and SCGSA methods were compared with the standard training algorithms (i.e., SCG, GD) and also with related techniques that might positively influence the network's robustness and generalization capabilities (e.g., learning internal representation [86] (GDIR), training with jitter [95] (ALG*), weight decay [119] (ALGWD) and learning from hints [106](ALG-*hint*)).

In this section, we use the notation described in Table 5.4 and in Subsection 5.1.3 on page 122. The generalization abilities of the trained BP-networks and their behavior on noisy data is indicated by the average value of the chosen error function on the test set (MSE_t, E_t) and on the noisy test set ($MSE(n_t)$, $E(n_t)$), respectively. In some of the experiments, we also compared the number of networks with no error on the test and noisy test sets (c, c_n) and the improvement of the error when compared to the SCG algorithm (imp).

The smoothness of the created network functions is indicated partly by the average values of absolute sensitivity coefficients on the test set (S_t) and by the average absolute values of weights and thresholds in the BP-network (w_m). The robustness of the training algorithms to noise in the data is further measured by the value of the error on the test data corrupted by varying amount of noise. In the following paragraphs, we will describe the settings and results of the experiments performed.

5.2.1 Experiment 5.2.1 – General results

Experiment setting

In Experiment 5.2.1 [81], we compared the SCGIR, SCGS and SCGSA training algorithms with pure scaled conjugate gradients (SCG) and other related techniques (SCG*, SCGIR*). We evaluated the generalization abilities of BP-networks trained by the above listed methods and their sensitivity to noise in the data.

SCG* and SCGIR* denote the SCG and SCGIR training algorithms combined with training with jitter [95]. See Section 3.4.3 on page 65 for a detailed description of this technique. For SCG* and SCGIR*, the training set was extended by noisy training patterns. The extended training set contained for each original training pattern also its copy corrupted by noise (0.01-1% of the original input values), the outputs remained the same. The chosen level of noise kept the input alterations within the same range like SCGSA. In the case of noisy test sets, however, larger amounts of added noise (5%) had been involved.

The tests involved two types of data: binary (*Binary Addition* and *Binary Multiplication* tasks – data sets **BIN2** and **BIN3**) and continuous (*World Bank* task – data set **WB**). These data sets contain several randomly generated input

features that made the training process more difficult. All methods were trained together with pruning of both input and hidden neurons (by the use of our training-and-pruning methodology described by Algorithms 4.3 and 4.4 on pages 89 and 90). The results for the **WB** data set were obtained by a 10-times repeated 10-fold cross-validation. Each tested method was applied to the same set of 100 different randomly initialized networks.

The parameters c_F and c_G of the SCGIR, SCGS and SCGSA methods were set experimentally, separately for each task. For the **BIN2** data set, all the trained networks had the initial topology 18-12-4, the parameter *maxEpochs* was set to 2001. For the **BIN3** data set, the initial networks' topologies were 18-12-12-6. The parameter *maxEpochs* was set to 1001 for the SCG* and SCGIR* methods, and to 601 for the other training algorithms. For the **WB** data set, we worked with two initial topologies: 35-50-5 and 35-15-15-5. For the network topology 35-50-5, the parameter *maxEpochs* was set to 1101, for the topology 35-15-15-5, it was set to 311.

The results obtained for the **BIN2** and **BIN3** data sets are stated in Table 5.5. Table 5.6 contains the results for the **WB** data set. Figure 5.1 depicts the histograms of $MSE(n_t)$ for the chosen training algorithms and all of the data sets. Figure 5.2 shows the average values of $MSE(n_t)$ for various levels of noise in the noisy test sets (for the chosen training algorithms and all of the data sets). In the tables and figures, the generalization abilities of the trained BP-networks and their sensitivity to noise in the data are indicated by the values of MSE_t, E_t, c and $MSE(n_t)$, $E(n_t)$, c_n, imp, respectively.

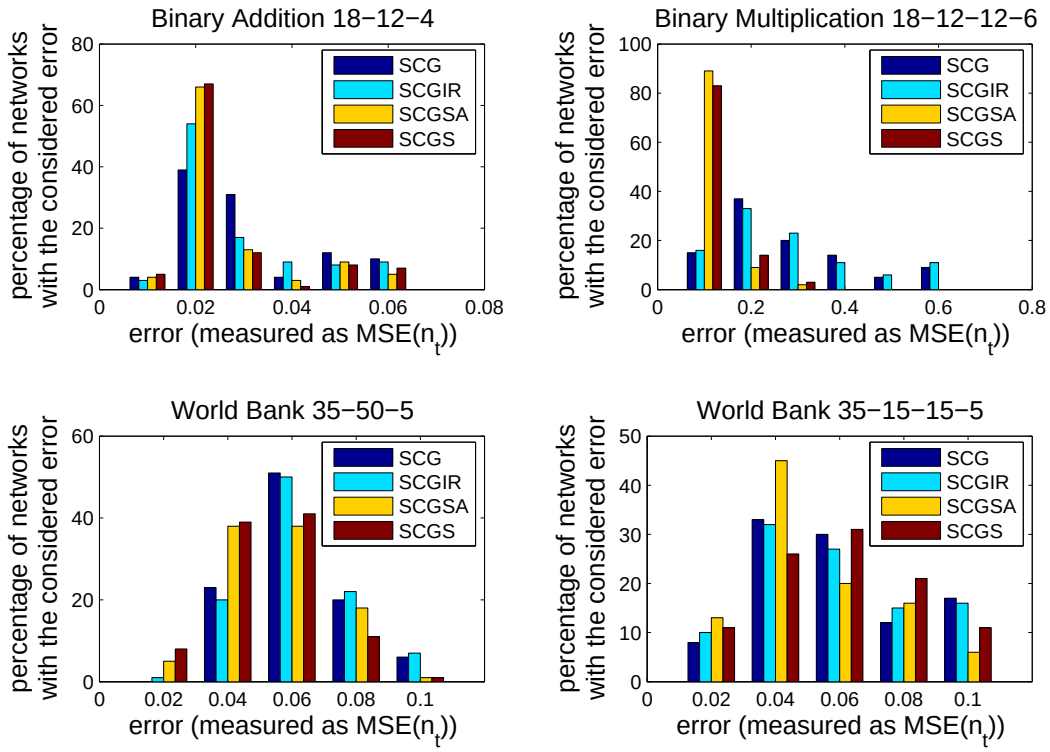


Figure 5.1: Experiment 5.2.1 – Histograms of $MSE(n_t)$ for the SCG, SCGIR, SCGS and SCGSA methods with pruning for the *Binary Addition*, *Binary Multiplication* and *World Bank* tasks (for networks with one and two hidden layers, respectively).

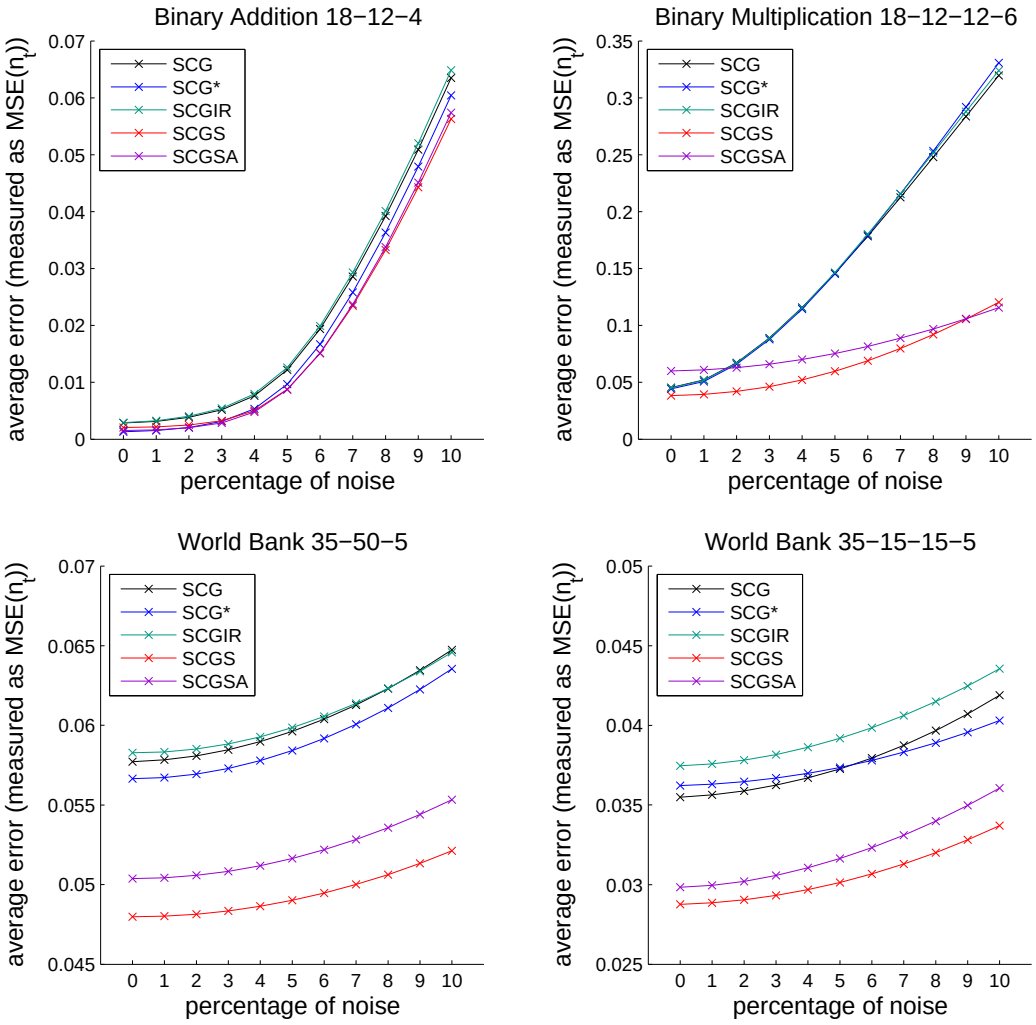


Figure 5.2: Experiment 5.2.1 – Average values of $MSE(n_t)$ for various noise levels and the SCG, SCG*, SCGIR, SCGS and SCGSA methods enhanced with pruning for the *Binary Addition*, *Binary Multiplication* and *World Bank* tasks (for networks with one and two hidden layers, respectively).

Results – On the generalization abilities of the networks

The experiments done on all datasets confirmed that both SCGS and SCGSA techniques significantly improve generalization capabilities of the trained networks. For the **BIN3** data set, the average value of $MSE(n_t)$ is for SCGS and SCGSA about 3.0-times smaller than for SCG, while for the **BIN2** and **WB** data sets, it is about 1.2-times smaller. The results for the SCGIR method are comparable with the SCG method. For the **BIN2** and **WB** data sets, the SCGS method outperformed all the other methods, while for the **BIN3** data set, the SCGSA method performed the best.

The SCGS and SCGSA methods remarkably reduce the overall network sensitivity indicated by the value of S_t (especially for the **WB** and **BIN3** data sets), while the average absolute value of weights (w_m) is comparable to SCG. These facts support the idea, that both SCGS and SCGSA methods favor smoother network functions. Not surprisingly, the exact SCGS method reduces the sensitivity

Table 5.5: Experiment 5.2.1 – The performance of the SCG, SCG*, SCGIR, SCGIR*, SCGSA and SCGS methods (with pruning) on the **BIN2** data set (*Binary Addition* task) and on the **BIN3** data set (*Binary Multiplication* task). The stated values correspond to the mean and standard deviation over 100 random network initializations.

Binary Addition task – using the 18-12-4 network topology

method	c_F	c_G	*arch*	n_I	n_A	c_n	c_R	c_A	p_{IR}	E_t	$E(n_t)$	*imp*	MSE_{tr}	MSE_t	$MSE(n_t)$	*imp*	S_t	w_m	epochs	t(s)
SCG	–	–	7.8-7.4	82	20	82	14	3	88.9	0.3 ± 1.7	5.0 ± 3.6	1.00	0.001 ± 0.006	0.003 ± 0.017	0.035 ± 0.028	1.00	0.09 ± 0.12	1.36 ± 0.43	1413.7	5.9
SCG*	–	–	6.4-7.4	91	23	87	25	13	90.4	0.1 ± 1.0	4.2 ± 3.7	1.21	0.001 ± 0.007	0.001 ± 0.008	0.031 ± 0.035	1.12	0.08 ± 0.09	1.44 ± 0.43	1312.9	7.1
SCGIR	10^{-5}	–	7.4-7.3	82	26	87	**67**	**25**	**96.7**	0.1 ± 0.6	4.9 ± 3.6	1.03	0.001 ± 0.007	0.001 ± 0.007	0.030 ± 0.025	1.18	0.07 ± 0.10	1.48 ± 0.51	1713.5	11.7
SCGIR	$10^{[-6,-4]}$	–	7.7-7.5	79	24	86	**52**	**22**	**95.2**	0.2 ± 1.7	5.4 ± 4.9	0.93	0.001 ± 0.012	0.002 ± 0.013	0.034 ± 0.054	1.02	0.09 ± 0.18	1.48 ± 0.54	1680.0	11.6
SCGIR	10^{-6}	–	7.6-7.5	84	25	85	39	18	93.0	0.3 ± 1.7	4.9 ± 3.4	1.03	0.001 ± 0.004	0.003 ± 0.015	0.036 ± 0.036	0.98	0.08 ± 0.11	1.41 ± 0.46	1429.2	11.2
SCGIR*	10^{-6}	–	6.8-7.3	89	24	90	29	11	92.2	0.0 ± 0.3	3.9 ± 3.1	1.29	0.001 ± 0.005	0.001 ± 0.005	0.031 ± 0.036	1.14	0.08 ± 0.08	1.41 ± 0.42	1384.3	15.6
SCGS	–	$5 \cdot 10^{-7}$	6.9-7.5	87	22	90	33	11	91.4	0.3 ± 1.7	4.0 ± 3.1	1.25	0.000 ± 0.003	0.002 ± 0.012	0.028 ± 0.022	1.27	0.07 ± 0.07	1.43 ± 0.49	1636.3	420.5
SCGS	–	$5 \cdot 10^{[-8,-6]}$	6.9-7.4	89	25	84	35	14	91.6	0.3 ± 1.8	4.4 ± 3.3	1.15	0.001 ± 0.004	0.003 ± 0.014	0.029 ± 0.026	1.20	0.08 ± 0.06	1.39 ± 0.44	1662.3	431.9
SCGS	10^{-6}	$5 \cdot 10^{-7}$	7.0-7.5	87	21	87	39	11	91.4	0.2 ± 1.2	4.0 ± 2.9	***1.27***	0.000 ± 0.003	0.002 ± 0.011	0.027 ± 0.020	***1.31***	0.07 ± 0.06	1.42 ± 0.50	1625.7	***420.1***
SCGS	10^{-6}	$5 \cdot 10^{[-8,-6]}$	7.1-7.5	86	24	87	38	14	91.7	0.1 ± 0.9	4.1 ± 2.3	***1.22***	0.000 ± 0.002	0.001 ± 0.007	0.027 ± 0.021	***1.29***	0.07 ± 0.06	1.40 ± 0.46	1750.7	***473.9***
SCGS	$10^{[-7,-5]}$	$5 \cdot 10^{-7}$	6.1-7.2	95	28	95	**43**	**16**	92.0	0.1 ± 0.6	4.0 ± 2.4	***1.27***	0.001 ± 0.005	0.001 ± 0.012	0.022 ± 0.016	***1.57***	0.07 ± 0.07	1.54 ± 0.50	1639.2	***351.6***
SCGS	$10^{[-7,-5]}$	$5 \cdot 10^{[-8,-6]}$	6.6-7.3	92	25	95	36	14	**92.1**	0.1 ± 0.8	4.1 ± 2.2	***1.23***	0.001 ± 0.004	0.001 ± 0.008	0.022 ± 0.011	***1.58***	0.07 ± 0.07	1.47 ± 0.48	1527.2	***271.8***
SCGSA	–	10^{-5}	6.9-7.5	90	22	86	29	12	90.4	0.1 ± 0.8	4.3 ± 3.1	1.16	0.001 ± 0.006	0.002 ± 0.009	0.028 ± 0.020	1.24	0.09 ± 0.09	1.35 ± 0.44	1308.6	8.4
SCGSA	–	$10^{[-6,-4]}$	7.1-7.6	84	21	83	32	13	90.3	0.2 ± 1.1	4.4 ± 2.3	1.15	0.001 ± 0.004	0.001 ± 0.008	0.029 ± 0.022	1.21	0.09 ± 0.07	1.33 ± 0.43	1292.5	10.2
SCGSA	10^{-6}	10^{-5}	7.4-7.8	84	19	87	31	10	90.3	0.4 ± 2.0	4.3 ± 3.5	**1.19**	0.001 ± 0.006	0.003 ± 0.015	0.029 ± 0.022	**1.23**	0.08 ± 0.08	1.32 ± 0.46	1378.3	**13.5**
SCGSA	10^{-6}	$10^{[-6,-4]}$	7.3-7.6	83	22	81	27	11	90.1	0.3 ± 2.1	4.6 ± 3.3	**1.11**	0.000 ± 0.001	0.003 ± 0.015	0.029 ± 0.024	**1.20**	0.09 ± 0.07	1.30 ± 0.44	1395.5	**16.7**
SCGSA	$10^{[-7,-5]}$	10^{-5}	6.4-7.3	90	24	94	**35**	**15**	**91.1**	0.1 ± 0.6	4.2 ± 2.5	**1.20**	0.001 ± 0.005	0.001 ± 0.007	0.024 ± 0.013	**1.47**	0.08 ± 0.08	1.42 ± 0.43	1363.0	**13.2**
SCGSA	$10^{[-7,-5]}$	$10^{[-6,-4]}$	6.4-7.3	93	24	96	31	14	**91.2**	0.1 ± 0.5	4.2 ± 2.5	**1.20**	0.001 ± 0.004	0.001 ± 0.004	0.023 ± 0.010	**1.50**	0.08 ± 0.06	1.42 ± 0.46	1341.0	**12.5**

Binary Multiplication task – using the 18-12-12-6 network topology

method	c_F	c_G	*arch*	n_I	c	c_n	p_{IR}	E_t	$E(n_t)$	*imp*	MSE_{tr}	MSE_t	$MSE(n_t)$	*imp*	S_t	w_m	epochs	t(s)
SCG	–	–	7.7-9.5-8.9	33	51	0	76.3	7.5 ± 9.5	25.3 ± 7.0	1.00	0.032 ± 0.040	0.045 ± 0.060	0.297 ± 0.189	1.00	0.28 ± 0.22	1.09 ± 0.21	601.0	6.1
SCG*	–	–	7.5-9.4-8.7	38	49	0	76.2	7.9 ± 9.6	25.4 ± 8.4	1.00	0.036 ± 0.041	0.044 ± 0.050	0.300 ± 0.210	0.99	0.26 ± 0.22	1.16 ± 0.28	1001.0	10.4
SCGIR	10^{-4}	–	7.5-9.4-8.8	36	46	0	**80.3**	6.9 ± 8.4	25.4 ± 7.2	0.99	0.032 ± 0.038	0.040 ± 0.054	0.343 ± 0.237	0.86	0.31 ± 0.28	1.21 ± 0.27	1001.0	11.4
SCGIR	$10^{[-5,-3]}$	–	7.6-9.5-8.8	35	48	0	**80.7**	9.9 ± 12.2	26.6 ± 8.0	0.95	0.046 ± 0.061	0.057 ± 0.079	0.338 ± 0.218	0.88	0.29 ± 0.26	1.18 ± 0.24	990.9	11.2
SCGIR	10^{-6}	–	7.6-9.6-9.0	35	50	0	76.2	7.1 ± 9.7	25.1 ± 7.5	1.01	0.031 ± 0.039	0.045 ± 0.063	0.301 ± 0.191	0.98	0.28 ± 0.23	1.08 ± 0.22	601.0	10.5
SCGIR*	10^{-6}	–	7.7-9.5-8.8	33	51	0	76.2	6.4 ± 8.5	24.7 ± 7.2	1.02	0.030 ± 0.038	0.041 ± 0.055	0.263 ± 0.145	1.13	0.27 ± 0.23	1.14 ± 0.24	1001.0	22.0
SCGS	–	$5 \cdot 10^{-7}$	7.6-9.6-8.9	34	55	11	72.6	6.5 ± 8.7	18.3 ± 8.7	1.38	0.029 ± 0.037	0.038 ± 0.048	0.098 ± 0.057	3.04	0.14 ± 0.09	0.95 ± 0.15	601.0	3733.8
SCGS	–	$5 \cdot 10^{[-8,-6]}$	7.5-9.5-8.9	38	53	9	73.3	6.8 ± 8.3	18.5 ± 8.6	1.37	0.030 ± 0.037	0.039 ± 0.046	0.110 ± 0.073	2.69	0.15 ± 0.11	0.98 ± 0.16	601.0	3688.1
SCGS	10^{-6}	$5 \cdot 10^{-7}$	7.8-9.5-8.9	31	53	10	72.7	6.1 ± 8.5	17.8 ± 8.4	***1.42***	0.028 ± 0.037	0.038 ± 0.050	0.097 ± 0.060	***3.05***	0.14 ± 0.09	0.96 ± 0.16	601.0	***3620.7***
SCGS	10^{-6}	$5 \cdot 10^{[-8,-6]}$	7.6-9.6-9.0	32	56	8	73.5	6.4 ± 7.8	17.7 ± 8.5	***1.42***	0.027 ± 0.033	0.040 ± 0.053	0.110 ± 0.084	***2.70***	0.16 ± 0.12	0.96 ± 0.15	601.0	***3723.8***
SCGS	$10^{[-7,-5]}$	$5 \cdot 10^{-7}$	7.6-9.6-9.0	36	57	11	73.1	4.7 ± 6.6	16.5 ± 7.7	***1.53***	0.023 ± 0.030	0.027 ± 0.034	0.088 ± 0.045	***3.37***	0.14 ± 0.11	0.98 ± 0.16	999.7	***3750.1***
SCGS	$10^{[-7,-5]}$	$5 \cdot 10^{[-8,-6]}$	7.6-9.6-9.0	35	56	15	73.4	5.7 ± 8.2	16.3 ± 8.5	***1.55***	0.024 ± 0.029	0.033 ± 0.044	0.098 ± 0.065	***3.04***	0.14 ± 0.13	0.96 ± 0.17	996.8	***3438.1***
SCGSA	–	$5 \cdot 10^{-5}$	7.9-9.6-9.0	30	38	19	72.4	9.6 ± 10.2	17.0 ± 9.7	1.49	0.040 ± 0.049	0.053 ± 0.061	0.085 ± 0.064	3.49	0.11 ± 0.06	0.89 ± 0.13	601.0	8.3
SCGSA	–	$5 \cdot 10^{[-6,-4]}$	7.7-9.6-8.8	34	32	4	73.7	11.1 ± 11.0	19.2 ± 10.1	1.32	0.043 ± 0.040	0.060 ± 0.066	0.101 ± 0.073	2.92	0.11 ± 0.09	0.94 ± 0.17	600.1	9.1
SCGSA	10^{-6}	$5 \cdot 10^{-5}$	7.9-9.7-8.9	27	42	16	72.5	8.6 ± 9.2	16.2 ± 8.9	**1.56**	0.034 ± 0.038	0.047 ± 0.049	0.079 ± 0.053	**3.77**	0.10 ± 0.06	0.88 ± 0.12	600.8	**13.3**
SCGSA	10^{-6}	$5 \cdot 10^{[-6,-4]}$	7.7-9.4-8.9	38	37	7	72.4	10.7 ± 10.9	19.0 ± 9.7	**1.33**	0.042 ± 0.046	0.059 ± 0.062	0.097 ± 0.064	**3.06**	0.11 ± 0.08	0.91 ± 0.14	599.1	**13.0**
SCGSA	$10^{[-7,-5]}$	$5 \cdot 10^{-5}$	7.6-9.4-8.9	40	45	18	71.2	8.5 ± 8.6	16.4 ± 8.9	**1.54**	0.036 ± 0.036	0.043 ± 0.042	0.074 ± 0.045	**3.99**	0.11 ± 0.07	0.90 ± 0.12	966.5	**12.9**
SCGSA	$10^{[-7,-5]}$	$5 \cdot 10^{[-6,-4]}$	7.4-9.5-9.0	49	42	13	72.9	9.1 ± 9.8	16.9 ± 9.5	**1.49**	0.038 ± 0.042	0.046 ± 0.048	0.080 ± 0.052	**3.69**	0.11 ± 0.07	0.92 ± 0.14	946.7	**13.7**

Table 5.6: Experiment 5.2.1 – The performance of the SCG, SCG*, SCGIR, SCGIR*, SCGSA and SCGS methods (with pruning) on the **WB** data set (*World Bank* task). The stated values correspond to the mean and standard deviation over 100 random network initializations.

World Bank task – using the 35-50-5 network topology																
method	c_F	c_G	*arch*	n_I	p_{IR}	E_t	$E(n_t)$	*imp*	MSE_{tr}	MSE_t	$MSE(n_t)$	*imp*	S_t	w_m	epochs	t(s)
SCG	–	–	22.2-24.8	97	65.5	0.029 ± 0.016	0.033 ± 0.017	1.00	0.058 ± 0.000	0.058 ± 0.017	0.063 ± 0.017	1.00	0.09 ± 0.03	0.36 ± 0.16	969.0	23.6
SCG*	–	–	22.1-23.5	96	70.5	0.031 ± 0.016	0.033 ± 0.017	1.00	0.057 ± 0.000	0.057 ± 0.017	0.062 ± 0.017	1.02	0.09 ± 0.03	0.39 ± 0.18	974.5	32.0
SCGIR	10^{-4}	–	24.3-17.8	71	**91.2**	0.041 ± 0.019	0.041 ± 0.018	0.80	0.066 ± 0.000	0.066 ± 0.020	0.070 ± 0.020	0.90	0.05 ± 0.02	0.48 ± 0.17	855.4	27.6
SCGIR	10^{-5}	–	23.1-23.7	88	**74.1**	0.032 ± 0.017	0.034 ± 0.018	0.96	0.059 ± 0.000	0.059 ± 0.018	0.063 ± 0.017	1.00	0.08 ± 0.03	0.37 ± 0.14	970.8	37.4
SCGIR	10^{-6}	–	23.1-25.5	93	64.8	0.030 ± 0.016	0.032 ± 0.018	1.03	0.058 ± 0.000	0.058 ± 0.017	0.063 ± 0.017	1.00	0.09 ± 0.03	0.34 ± 0.15	968.0	48.5
SCGIR	$10^{[-7,-5]}$	–	22.1-24.3	95	67.9	0.030 ± 0.018	0.033 ± 0.017	1.01	0.057 ± 0.000	0.057 ± 0.017	0.062 ± 0.017	1.01	0.09 ± 0.03	0.38 ± 0.19	1014.8	41.7
SCGIR*	10^{-6}	–	22.6-24.0	93	70.2	0.029 ± 0.016	0.034 ± 0.018	0.97	0.057 ± 0.000	0.057 ± 0.017	0.062 ± 0.018	1.01	0.08 ± 0.03	0.37 ± 0.13	967.6	78.2
SCGS	–	10^{-5}	22.4-21.9	89	73.2	0.028 ± 0.017	0.032 ± 0.017	1.03	0.048 ± 0.000	0.048 ± 0.015	0.051 ± 0.016	1.23	0.05 ± 0.02	0.38 ± 0.15	964.0	2036.0
SCGS	–	$10^{[-6,-4]}$	23.6-24.5	82	67.2	0.029 ± 0.017	0.031 ± 0.016	1.07	0.052 ± 0.000	0.052 ± 0.015	0.054 ± 0.015	1.16	0.05 ± 0.02	0.33 ± 0.16	955.3	2050.3
SCGS	10^{-6}	10^{-5}	22.7-22.0	88	**75.3**	0.030 ± 0.018	0.032 ± 0.019	***1.04***	0.048 ± 0.000	0.048 ± 0.015	0.050 ± 0.016	***1.25***	0.05 ± 0.02	0.39 ± 0.14	968.4	***1984.2***
SCGS	10^{-6}	$10^{[-6,-4]}$	24.1-23.7	80	**70.2**	0.028 ± 0.016	0.030 ± 0.016	***1.08***	0.051 ± 0.000	0.051 ± 0.016	0.054 ± 0.016	***1.17***	0.05 ± 0.02	0.34 ± 0.14	952.0	***1951.3***
SCGS	$10^{[-7,-5]}$	10^{-5}	23.9-22.4	66	**76.4**	0.028 ± 0.017	0.031 ± 0.016	***1.06***	0.048 ± 0.000	0.048 ± 0.016	0.051 ± 0.016	***1.23***	0.04 ± 0.02	0.38 ± 0.14	985.9	***1819.9***
SCGS	$10^{[-7,-5]}$	$10^{[-6,-4]}$	24.5-23.9	59	**73.8**	0.028 ± 0.014	0.031 ± 0.015	***1.07***	0.051 ± 0.000	0.051 ± 0.016	0.054 ± 0.016	***1.16***	0.05 ± 0.02	0.37 ± 0.18	987.0	***1704.2***
SCGSA	–	$2 \cdot 10^{-4}$	22.8-22.9	97	70.6	0.028 ± 0.018	0.031 ± 0.018	1.06	0.050 ± 0.000	0.050 ± 0.016	0.054 ± 0.015	1.17	0.06 ± 0.03	0.37 ± 0.13	978.2	35.2
SCGSA	–	$2 \cdot 10^{[-5,-3]}$	23.0-22.6	93	70.5	0.031 ± 0.018	0.034 ± 0.018	0.98	0.054 ± 0.000	0.054 ± 0.018	0.057 ± 0.018	1.10	0.06 ± 0.03	0.39 ± 0.18	962.3	35.1
SCGSA	10^{-6}	$2 \cdot 10^{-4}$	22.4-21.6	97	**73.3**	0.028 ± 0.018	0.031 ± 0.018	**1.07**	0.051 ± 0.000	0.051 ± 0.017	0.055 ± 0.017	**1.14**	0.06 ± 0.02	0.41 ± 0.19	945.4	**59.0**
SCGSA	10^{-6}	$2 \cdot 10^{[-5,-3]}$	23.2-23.5	92	**70.2**	0.030 ± 0.019	0.031 ± 0.017	**1.05**	0.053 ± 0.000	0.053 ± 0.016	0.056 ± 0.016	**1.12**	0.06 ± 0.03	0.37 ± 0.13	965.2	**58.0**
SCGSA	$10^{[-7,-5]}$	$2 \cdot 10^{-4}$	22.9-24.1	89	**72.4**	0.028 ± 0.016	0.032 ± 0.017	**1.02**	0.052 ± 0.000	0.052 ± 0.016	0.056 ± 0.016	**1.12**	0.07 ± 0.02	0.37 ± 0.13	978.8	**52.4**
SCGSA	$10^{[-7,-5]}$	$2 \cdot 10^{[-5,-3]}$	23.1-23.8	86	**71.2**	0.032 ± 0.019	0.033 ± 0.016	1.00	0.055 ± 0.001	0.055 ± 0.016	0.058 ± 0.016	**1.09**	0.07 ± 0.03	0.37 ± 0.13	990.6	**48.3**

World Bank task – using the 35-15-15-5 network topology																
method	c_F	c_G	*arch*	n_I	p_{IR}	E_t	$E(n_t)$	*imp*	MSE_{tr}	MSE_t	$MSE(n_t)$	*imp*	S_t	w_m	epochs	t(s)
SCG	–	–	21.2-11.2-7.3	88	78.2	0.031 ± 0.017	0.036 ± 0.020	1.00	0.027 ± 0.037	0.061 ± 0.037	0.064 ± 0.038	1.00	0.05 ± 0.02	0.52 ± 0.12	305.1	15.6
SCG*	–	–	21.0-11.4-7.1	72	79.1	0.031 ± 0.017	0.039 ± 0.021	0.94	0.021 ± 0.019	0.055 ± 0.025	0.057 ± 0.025	1.11	0.04 ± 0.02	0.47 ± 0.10	305.9	18.5
SCGIR	10^{-4}	–	21.8-10.8-7.5	76	**87.2**	0.037 ± 0.020	0.044 ± 0.022	0.83	0.027 ± 0.021	0.061 ± 0.029	0.065 ± 0.029	0.98	0.05 ± 0.03	0.57 ± 0.14	302.3	18.8
SCGIR	10^{-5}	–	22.0-11.6-7.8	82	**80.6**	0.033 ± 0.017	0.039 ± 0.018	0.93	0.023 ± 0.022	0.058 ± 0.026	0.061 ± 0.026	1.04	0.04 ± 0.02	0.52 ± 0.14	303.6	20.4
SCGIR	10^{-6}	–	21.1-11.7-7.4	90	77.6	0.031 ± 0.017	0.036 ± 0.020	1.02	0.030 ± 0.037	0.061 ± 0.037	0.064 ± 0.037	0.99	0.05 ± 0.02	0.50 ± 0.11	304.5	24.2
SCGIR	$10^{[-7,-5]}$	–	20.3-11.3-7.3	90	78.6	0.032 ± 0.017	0.038 ± 0.020	0.97	0.023 ± 0.022	0.057 ± 0.028	0.061 ± 0.032	1.05	0.05 ± 0.02	0.52 ± 0.11	305.8	20.7
SCGIR*	10^{-6}	–	21.4-11.5-7.4	76	79.5	0.032 ± 0.018	0.039 ± 0.020	0.94	0.025 ± 0.028	0.060 ± 0.029	0.062 ± 0.029	1.03	0.04 ± 0.02	0.49 ± 0.10	301.9	33.8
SCGS	–	10^{-5}	21.8-11.8-7.5	89	***79.6***	0.029 ± 0.018	0.032 ± 0.019	***1.14***	0.021 ± 0.015	0.050 ± 0.021	0.050 ± 0.021	***1.26***	0.03 ± 0.01	0.53 ± 0.13	302.7	***53188.0***
SCGS	–	$10^{[-6,-4]}$	22.6-11.9-7.6	92	***78.1***	0.032 ± 0.019	0.029 ± 0.019	***1.26***	0.032 ± 0.037	0.060 ± 0.033	0.049 ± 0.034	***1.30***	0.03 ± 0.02	0.52 ± 0.12	304.4	***54040.0***
SCGSA	–	10^{-4}	21.0-11.7-7.4	94	78.6	0.029 ± 0.016	0.034 ± 0.017	**1.08**	0.021 ± 0.017	0.050 ± 0.021	0.052 ± 0.022	**1.23**	0.04 ± 0.02	0.51 ± 0.10	303.4	**19.8**
SCGSA	–	$10^{[-5,-3]}$	21.6-11.8-7.5	89	78.0	0.028 ± 0.017	0.036 ± 0.019	**1.02**	0.023 ± 0.028	0.053 ± 0.029	0.055 ± 0.029	**1.15**	0.04 ± 0.02	0.50 ± 0.12	304.6	**19.3**
SCGSA	10^{-6}	10^{-4}	21.1-11.4-7.0	95	78.6	0.029 ± 0.016	0.035 ± 0.016	1.04	0.022 ± 0.023	0.052 ± 0.025	0.054 ± 0.025	1.19	0.04 ± 0.02	0.51 ± 0.10	304.1	28.5
SCGSA	10^{-6}	$10^{[-5,-3]}$	21.5-11.7-7.5	89	78.5	0.030 ± 0.017	0.034 ± 0.018	1.07	0.027 ± 0.033	0.056 ± 0.032	0.058 ± 0.031	1.11	0.04 ± 0.02	0.50 ± 0.13	304.7	27.6
SCGSA	$10^{[-7,-5]}$	10^{-4}	21.1-11.7-7.2	84	78.5	0.030 ± 0.017	0.036 ± 0.019	1.01	0.025 ± 0.047	0.056 ± 0.041	0.058 ± 0.042	1.09	0.04 ± 0.02	0.51 ± 0.11	304.6	24.1

more efficiently than the approximative SCGSA method for most of the data sets (except **BIN3**, for which the SCGSA-trained networks generalize even slightly better).

The SCG* and SCGIR* methods don't improve the generalization ability of the trained BP-networks as significantly as SCGS and SCGSA (as indicated, e.g., by $MSE(n_t)$ and imp in Tables 5.5 and 5.6). For some of the tasks (**BIN3** and **WB** with the topology 35-50-5), SCG* and SCGIR* don't even outperform SCG.

Results – On the robustness to the level of noise in the data

In Figure 5.2, we can clearly see, that both the SCGS and SCGSA methods are remarkably stable when tested on data corrupted by various amounts of noise. The SCG* and SCGIR methods are less stable when compared to SCGS and SCGSA, their robustness to noise is similar to SCG.

5.2.2 Experiment 5.2.2 – Extended results

Experiment setting

In Experiment 5.2.2 [83, 84], we extended the setting of Experiment 5.2.1 on BP-networks trained without pruning or with pruning of only hidden or only input neurons. We compared the SCGIR, SCGS and SCGSA methods with the standard SCG training algorithm and with the variant of the SCGS method, where the sensitivity is enforced ($c_G < 0$, we denote this method as SCGS-ES). The sensitivity enforcement is assumed to increase the differences among the achieved sensitivity coefficients of the respective neurons and restrict the space of candidate hypotheses for the wanted network function [83].

The tests involved two types of data: binary (*Binary Addition* and *Binary Multiplication* tasks – data sets **BIN2** and **BIN3**) and continuous (*World Bank* task – data set **WB**). The experiment setting is analogical to Experiment 5.2.1 on page 123 with the following exceptions: For the **BIN2** data set, the parameter $maxEpochs$ was set to 5001 and the tests comprise more possible values of the parameters c_F and c_G. The results for the **WB** data set were obtained by 10-fold cross-validation. For the **BIN2** and **BIN3** data sets, each tested method was repeated 100-times on 100 different randomly initialized BP-networks.

Tables 5.7 and 5.8 contain the results for the **BIN2** data set, while Tables 5.9 and 5.11 summarize the experiment on the **WB** data set. The results obtained for the **BIN3** data set are stated in Table 5.10. In the tables, the generalization abilities of the trained BP-networks and their sensitivity to noise in the data are indicated by the values of MSE_t, E_t, c and $MSE(n_t)$, $E(n_t)$, c_n, S_t, imp, respectively.

Results – On the generalization abilities of the networks

All experiments confirmed that the both the SCGS and SCGSA methods significantly improve generalization capabilities while maintaining a relatively stable behavior on noisy data even when trained without pruning or with pruning of just input or just hidden neurons. Sensitivity enforcement (SCGS-ES) showed in this respect only marginal improvements.

Table 5.7: Experiment 5.2.2 – The performance of the SCG, SCGIR, SCGS-ES, SCGS and SCGSA methods without pruning on the **BIN2** data set using the 18-12-4 network topology. The stated values correspond to the mean and standard deviation over 100 random network initializations.

method	c_F	c_G	E_{tr}	E_v	E_t	c	epochs	t(s)
SCG	–	–	3.05 ± 11.14	4.38 ± 9.92	4.60 ± 10.36	76	342.1	1.1
SCGIR	$2 \cdot 10^{-6}$	–	3.04 ± 11.21	4.32 ± 9.96	4.36 ± 10.29	76	357.1	2.9
SCGIR	$2 \cdot 10^{-5}$	–	3.04 ± 11.26	4.39 ± 10.06	4.31 ± 10.06	79	638.3	4.6
SCGIR	$5 \cdot 10^{-5}$	–	3.15 ± 10.83	4.43 ± 9.35	4.77 ± 10.44	74	625.9	5.1
SCGIR	$5 \cdot 10^{-4}$	–	6.6 ± 20.55	8.46 ± 12.4	8.15 ± 12.15	44	1271.0	10.4
SCGS-ES	–	-10^{-7}	2.94 ± 11.09	4.20 ± 9.85	4.21 ± 10.04	75	333.9	107.2
SCGS-ES	–	$-2 \cdot 10^{-7}$	2.94 ± 11.09	4.20 ± 9.85	4.18 ± 10.04	77	409.5	141.6
SCGS-ES	–	-10^{-6}	3.58 ± 11.99	5.46 ± 10.69	5.52 ± 11.14	64	329.8	105.5
SCGS-ES	–	$-2 \cdot 10^{-6}$	4.19 ± 13.22	6.83 ± 11.11	6.39 ± 11.38	27	181.4	65.0
SCGS-ES	$2 \cdot 10^{-5}$	$-2 \cdot 10^{-7}$	3.15 ± 10.74	4.75 ± 9.47	5.03 ± 10.57	71	558.8	181.0
SCGS-ES	$5 \cdot 10^{-4}$	$-2 \cdot 10^{-7}$	6.83 ± 20.58	8.91 ± 12.42	8.65 ± 12.42	42	826.8	269.8
SCGS	–	10^{-4}	0.94 ± 2.53	2.49 ± 3.21	3.02 ± 3.30	19	557.8	216.7
SCGS	–	$2 \cdot 10^{-4}$	5.28 ± 5.00	6.91 ± 3.52	7.33 ± 3.93	1	220.9	55.3
SCGS	–	$2 \cdot 10^{-5}$	$\mathbf{0.09 \pm 0.51}$	$\mathbf{0.69 \pm 2.54}$	$\mathbf{0.73 \pm 2.76}$	**86**	749.4	229.0
SCGS	–	$2 \cdot 10^{-6}$	2.47 ± 9.25	3.04 ± 8.52	3.15 ± 8.72	84	432.6	127.4
SCGS	–	$2 \cdot 10^{-7}$	2.96 ± 11.18	3.95 ± 9.55	4.02 ± 9.94	79	312.9	107.3
SCGS	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	0.15 ± 0.80	0.83 ± 2.71	0.89 ± 3.07	83	1248.8	328.2
SCGS	$5 \cdot 10^{-4}$	$2 \cdot 10^{-5}$	1.84 ± 7.15	3.52 ± 7.56	3.69 ± 7.83	57	954.4	372.1
SCGSA	–	$5 \cdot 10^{-4}$	0.76 ± 3.37	1.70 ± 4.98	$\mathbf{1.75 \pm 5.15}$	66	788.5	5.0
SCGSA	–	$2 \cdot 10^{-4}$	1.03 ± 4.93	1.89 ± 6.22	2.02 ± 6.57	77	880.0	5.6
SCGSA	–	10^{-4}	1.48 ± 6.09	2.64 ± 7.62	2.64 ± 7.77	80	758.8	4.8
SCGSA	–	$8 \cdot 10^{-5}$	1.47 ± 5.83	2.61 ± 7.67	2.57 ± 7.63	**81**	681.7	4.4
SCGSA	$2 \cdot 10^{-6}$	$5 \cdot 10^{-4}$	$\mathbf{0.73 \pm 3.35}$	$\mathbf{1.61 \pm 4.92}$	$\mathbf{1.77 \pm 5.28}$	63	641.0	6.6
SCGSA	$2 \cdot 10^{-6}$	$2 \cdot 10^{-4}$	0.86 ± 4.69	1.61 ± 5.56	1.79 ± 6.21	80	835.9	8.8
SCGSA	$2 \cdot 10^{-6}$	10^{-4}	1.48 ± 6.03	2.62 ± 7.67	2.56 ± 7.72	**81**	877.8	9.2
The **BIN2A** dataset - using the topology 6-6-4								
SCG	–	–	20.07 ± 27.74	6.69 ± 9.25	6.69 ± 9.25	55	4852.2	15.5
SCGS	–	$2 \cdot 10^{-5}$	13.68 ± 24.50	4.56 ± 8.17	4.56 ± 8.17	58	2329.9	215.8
SCGSA	10^{-6}	10^{-6}	13.14 ± 24.42	4.38 ± 8.14	4.38 ± 8.14	61	2224.4	15.6
The **BIN2A** dataset - using the topology 6-12-4								
SCG	–	–	0 ± 0	0 ± 0	0 ± 0	100	1000.0	4.1

While the SCGS and SCGSA methods achieve relatively similar results for networks trained with pruning of both input and hidden neurons, if pruning of input neurons is not allowed, the SCGS method outperforms SCGSA in the generalization abilities of the trained BP-networks, especially for the **BIN2** and **BIN3** data sets (as indicated, e.g., by the value of $MSE(n_t)$ and *imp* in Table 5.10).

In Tables 5.7 and 5.8, we can clearly see that the *Binary Addition* task is harder to be solved by a neural network when random inputs are added and no pruning of inputs is done. In such a case, both the SCGS and SCGSA methods remarkably increase the number of BP-networks that learned the task (indicated by the value of c). When comparing the SCGS method to SCG, the average number of erroneously recalled input patterns over all of the 100 trained networks (E_t) has been reduced 6-times while reducing its standard deviation 4-times. The number c of error-less networks has raised by 13%. For the SCGSA method compared to SCG, the value of E_t has been reduced 2.5-times while reducing its standard deviation 2-times. The number of error-less networks has raised by 7%. If pruning of inputs was allowed, almost all BP-networks learned the task correctly, independently of the chosen training algorithm.

Also the task of *Binary Multiplication* is hard to be solved by a BP-network when random inputs are added and no pruning of inputs is done, as indicated

Table 5.8: Experiment 5.2.2 – The performance of the SCG, SCGIR, SCGS-ES, SCGS and SCGSA methods with pruning on the **BIN2** data set using the 18-12-4 network topology. The stated values correspond to the mean and standard deviation over 100 random network initializations.

method	c_F	c_G	H	I	n_H	n_I	E_{tr}	E_v	E_t	c	epochs	t(s)
Pruning of hidden neurons												
SCG	–	–	8.17 ± 2.12	18	23	–	2.79 ± 11.99	2.71 ± 7.84	3.03 ± 8.52	80	647.2	2.1
SCGIR	$2 \cdot 10^{-5}$	–	7.78 ± 2.00	18	32	–	3.71 ± 13.51	2.64 ± 7.28	2.98 ± 8.42	85	948.3	7.3
SCGS-ES	–	$-2 \cdot 10^{-7}$	7.96 ± 2.07	18	28	–	2.66 ± 11.96	2.54 ± 7.78	2.75 ± 8.41	83	744.8	221.6
SCGS-ES	$2 \cdot 10^{-5}$	$-2 \cdot 10^{-7}$	7.97 ± 2.12	18	30	–	3.27 ± 12.60	2.49 ± 6.93	2.73 ± 7.87	84	985.0	251.3
SCGS	–	$2 \cdot 10^{-5}$	7.75 ± 2.15	18	**38**	–	$\mathbf{0.18 \pm 0.89}$	$\mathbf{0.49 \pm 1.92}$	$\mathbf{0.66 \pm 2.72}$	**91**	1138.0	327.1
SCGS	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	7.77 ± 2.15	18	37	–	0.37 ± 2.00	0.82 ± 3.13	0.79 ± 2.98	88	1522.6	419.2
SCGSA	–	$5 \cdot 10^{-5}$	7.61 ± 1.91	18	**32**	–	$\mathbf{1.32 \pm 7.24}$	$\mathbf{1.29 \pm 5.10}$	$\mathbf{1.33 \pm 5.31}$	**89**	1021.3	6.3
SCGSA	$2 \cdot 10^{-6}$	$5 \cdot 10^{-5}$	8.25 ± 2.31	18	27	–	1.90 ± 9.93	1.87 ± 6.99	1.86 ± 6.77	88	1040.2	10.4
SCGS	–	$2 \cdot 10^{-4}$	7.73 ± 2.05	18	37	-	3.59 ± 3.80	3.72 ± 3.05	4.11 ± 3.33	13	436.9	96.6
SCGS	–	$1 \cdot 10^{-4}$	7.96 ± 2.25	18	34	-	0.67 ± 2.49	1.23 ± 2.1	1.59 ± 2.08	31	957.3	328.6
SCGS	–	$2 \cdot 10^{-6}$	7.86 ± 2.02	18	27	-	2.31 ± 10.73	1.87 ± 6.89	1.87 ± 6.85	86	758.6	206.1
SCGS	–	$2 \cdot 10^{-7}$	8.27 ± 2.21	18	23	-	2.62 ± 11.71	2.36 ± 7.44	2.65 ± 8.07	83	616.2	185.3
Pruning of input neurons												
SCG	–	–	12	6.67 ± 2.63	–	89	0 ± 0	0 ± 0	0 ± 0	100	1339.1	5.3
SCGIR	$2 \cdot 10^{-5}$	–	12	6.20 ± 1.26	–	94	0 ± 0	0 ± 0	0 ± 0	100	1029.2	8.7
SCGS-ES	–	$-2 \cdot 10^{-7}$	12	6.76 ± 2.86	–	91	0 ± 0	0 ± 0	0 ± 0	100	1114.4	215.1
SCGS-ES	$2 \cdot 10^{-5}$	$-2 \cdot 10^{-7}$	12	6.55 ± 2.37	–	91	0 ± 0	0 ± 0	0 ± 0	100	963.0	199.7
SCGS	–	$2 \cdot 10^{-5}$	12	$\mathbf{6.00 \pm 0.00}$	–	**100**	0 ± 0	0 ± 0	0 ± 0	100	1148.1	282.3
SCGS	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	12	$\mathbf{6.00 \pm 0.00}$	–	**100**	0 ± 0	0 ± 0	0 ± 0	100	1447.8	339.0
SCGSA	–	10^{-5}	12	$\mathbf{6.00 \pm 0.00}$	–	**100**	0 ± 0	0 ± 0	0 ± 0	100	977.7	7.2
SCGSA	$2 \cdot 10^{-6}$	10^{-5}	12	6.01 ± 0.10	–	99	0 ± 0	0 ± 0	0 ± 0	100	939.7	10.5
SCGS	–	$2 \cdot 10^{-4}$	12	6.02 ± 0.2	–	99	3.24 ± 6.03	1.1 ± 2.06	1.13 ± 2.16	62	432.9	77.6
SCGS	–	$1 \cdot 10^{-4}$	12	6.0 ± 0.0	–	100	0.24 ± 1.1	0.08 ± 0.37	0.08 ± 0.37	95	745.9	227.8
SCGS	–	$2 \cdot 10^{-6}$	12	6.02 ± 0.2	–	99	0 ± 0	0 ± 0	0 ± 0	100	1043.6	197.1
SCGS	–	$2 \cdot 10^{-7}$	12	6.04 ± 0.24	–	97	0 ± 0	0 ± 0	0 ± 0	100	1173.2	200
Pruning of both input and hidden neurons												
SCG	–	–	7.74 ± 1.64	7.22 ± 3.49	23	85	0.01 ± 0.1	0.09 ± 0.90	0.13 ± 1.30	99	1343.1	4.5
SCGIR	$2 \cdot 10^{-5}$	–	7.40 ± 1.41	6.91 ± 3.08	29	88	0.03 ± 0.30	0.01 ± 0.10	0.01 ± 0.10	99	1583.5	11.4
SCGS-ES	–	$-2 \cdot 10^{-7}$	7.65 ± 1.56	7.42 ± 3.77	23	84	0 ± 0	0 ± 0	0 ± 0	100	1339.0	258.2
SCGS-ES	$2 \cdot 10^{-5}$	$-2 \cdot 10^{-7}$	7.56 ± 1.44	7.29 ± 3.61	23	84	0.15 ± 1.23	0.05 ± 0.41	0.05 ± 0.41	98	1506.4	284.9
SCGS	–	$2 \cdot 10^{-5}$	$\mathbf{6.98 \pm 1.24}$	$\mathbf{6.52} \pm 2.38$	**43**	**94**	0 ± 0	0 ± 0	0 ± 0	100	1597.1	343.2
SCGS	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	$\mathbf{7.06 \pm 1.24}$	$\mathbf{6.50 \pm 2.37}$	**37**	**95**	0.07 ± 0.70	0.21 ± 2.10	0.19 ± 1.90	99	1850.3	451.5
SCGSA	–	$5 \cdot 10^{-5}$	$\mathbf{7.08 \pm 1.16}$	$\mathbf{6.44 \pm 2.09}$	**35**	**92**	0 ± 0	0 ± 0	0 ± 0	100	1246.2	8.1
SCGSA	$2 \cdot 10^{-6}$	$5 \cdot 10^{-5}$	7.22 ± 1.28	6.44 ± 2.09	30	93	0 ± 0	0 ± 0	0 ± 0	100	1352.0	13.6
SCGS	–	$2 \cdot 10^{-4}$	7.03 ± 1.11	6.14 ± 1.21	37	98	2.33 ± 8.24	0.77 ± 2.7	0.82 ± 2.79	82	558.3	108.3
SCGS	–	$1 \cdot 10^{-4}$	7.02 ± 1.26	6.61 ± 2.63	41	94	0.15 ± 1.23	0.09 ± 0.57	0.1 ± 0.64	97	1154.2	296.5
SCGS	–	$2 \cdot 10^{-6}$	7.36 ± 1.33	6.27 ± 1.33	29	91	0 ± 0	0 ± 0	0 ± 0	100	1384.9	252.9
SCGS	–	$2 \cdot 10^{-7}$	7.34 ± 1.23	6.07 ± 0.43	27	96	0 ± 0	0 ± 0	0 ± 0	100	1300	230.8
Minimal 6-6-4 topology without pruning												
SCGS	–	$2 \cdot 10^{-5}$	6	6	–	–	20.07 ± 27.74	6.69 ± 9.25	6.69 ± 9.25	55	4852.2	15.5
SCGS	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	6	6	–	–	13.68 ± 24.50	4.56 ± 8.17	4.56 ± 8.17	58	2329.9	215.8
SCGSA	–	10^{-6}	6	6	–	–	14.70 ± 26.24	4.90 ± 8.75	4.90 ± 8.75	**65**	2350.5	10.4
SCGSA	$2 \cdot 10^{-6}$	10^{-6}	6	6	–	–	13.14 ± 24.42	4.38 ± 8.14	4.38 ± 8.14	61	2224.4	15.6

Table 5.9: Experiment 5.2.2 - The performance of the SCG, SCGIR, SCGS, SCGS-ES and SCGSA methods with and without pruning on the **WB** data set using the 35-50-5 network topology and 10-fold cross-validation.

method	c_F	c_G	H	I	E_{tr}	E_v	E_t	epochs	t(s)
Without pruning									
SCG	–	–	50	35	0.002 ± 0.002	0.019 ± 0.018	0.041 ± 0.020	115.3 ± 36.2	2.7 ± 1.1
SCGIR	10^{-6}	–	50	35	0.002 ± 0.002	0.019 ± 0.018	0.044 ± 0.021	115.8 ± 36.7	8.0 ± 2.5
SCGS	–	$2 \cdot 10^{-5}$	50	35	0.001 ± 0.001	0.017 ± 0.014	$\mathbf{0.034 \pm 0.014}$	131.1 ± 29.8	452.5 ± 155.9
SCGS	10^{-6}	$2 \cdot 10^{-5}$	50	35	0.001 ± 0.001	0.017 ± 0.014	$\mathbf{0.035 \pm 0.015}$	131.1 ± 30.6	399.1 ± 91.2
SCGS-ES	–	$-2 \cdot 10^{-6}$	50	35	0.003 ± 0.003	0.019 ± 0.015	0.044 ± 0.022	110.2 ± 36.9	349.6 ± 118.8
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	50	35	0.002 ± 0.003	0.022 ± 0.020	0.043 ± 0.022	110.4 ± 37.6	340.9 ± 117.3
SCGSA	–	10^{-3}	50	35	0.003 ± 0.002	0.016 ± 0.013	$\mathbf{0.033 \pm 0.016}$	139.0 ± 21.2	$\mathbf{6.1 \pm 1.0}$
SCGSA	10^{-6}	10^{-3}	50	35	0.003 ± 0.003	0.019 ± 0.016	$\mathbf{0.033 \pm 0.015}$	117.1 ± 31.0	$\mathbf{10.5 \pm 2.7}$
Pruning of hidden neurons									
SCG	–	–	41.9 ± 3.3	35	0.001 ± 0.001	0.026 ± 0.012	0.037 ± 0.021	280.5 ± 54.4	6.3 ± 1.2
SCGIR	10^{-6}	–	28.9 ± 12.2	35	0.001 ± 0.001	0.025 ± 0.012	0.034 ± 0.021	387.9 ± 110.9	20.3 ± 4.6
SCGS	–	$2 \cdot 10^{-5}$	34.0 ± 10.5	35	0.001 ± 0.001	0.015 ± 0.009	$\mathbf{0.031 \pm 0.011}$	447.3 ± 187.4	1326.2 ± 512.4
SCGS	10^{-6}	$2 \cdot 10^{-5}$	33.8 ± 11.5	35	0.003 ± 0.004	0.017 ± 0.012	$\mathbf{0.032 \pm 0.011}$	457.3 ± 161.8	1227.2 ± 403.8
SCGS-ES	–	$-2 \cdot 10^{-6}$	33.8 ± 10.5	35	0.002 ± 0.003	0.030 ± 0.016	0.033 ± 0.016	288.7 ± 84.0	792.9 ± 227.0
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	33.7 ± 10.2	35	0.001 ± 0.001	0.032 ± 0.017	0.035 ± 0.018	302.6 ± 91.8	854.4 ± 253.3
SCGSA	–	10^{-5}	$\mathbf{21.8 \pm 9.4}$	35	0.003 ± 0.003	0.023 ± 0.008	$\mathbf{0.029 \pm 0.017}$	552.6 ± 236.5	$\mathbf{14.3 \pm 4.5}$
SCGSA	10^{-6}	10^{-5}	$\mathbf{24.1 \pm 10.1}$	35	0.002 ± 0.002	0.019 ± 0.012	$\mathbf{0.027 \pm 0.010}$	599.4 ± 182.9	$\mathbf{29.6 \pm 8.0}$
Pruning of input neurons									
SCG	–	–	50	22.7 ± 5.1	0.009 ± 0.005	0.009 ± 0.009	0.028 ± 0.017	294.7 ± 79.0	9.9 ± 2.3
SCGIR	10^{-6}	–	50	22.2 ± 2.0	0.009 ± 0.003	0.016 ± 0.010	0.035 ± 0.022	267.9 ± 97.4	20.3 ± 6.6
SCGS	–	$2 \cdot 10^{-5}$	50	$\mathbf{21.1 \pm 2.6}$	0.009 ± 0.003	0.015 ± 0.009	$\mathbf{0.024 \pm 0.020}$	308.7 ± 94.2	893.8 ± 286.3
SCGS	10^{-6}	$2 \cdot 10^{-5}$	50	$\mathbf{21.2 \pm 2.1}$	0.008 ± 0.004	0.017 ± 0.010	$\mathbf{0.025 \pm 0.019}$	316.8 ± 105.8	787.1 ± 247.5
SCGS-ES	–	$-2 \cdot 10^{-6}$	50	21.6 ± 3.2	0.012 ± 0.003	0.016 ± 0.008	0.037 ± 0.023	257.1 ± 89.5	592.8 ± 198.5
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	50	21.6 ± 3.2	0.013 ± 0.003	0.018 ± 0.010	0.035 ± 0.020	257.3 ± 88.7	590.3 ± 195.5
SCGSA	–	10^{-4}	50	21.5 ± 2.9	0.009 ± 0.002	0.017 ± 0.010	0.029 ± 0.020	239.8 ± 97.7	$\mathbf{12.1 \pm 4.4}$
SCGSA	10^{-6}	10^{-4}	50	$\mathbf{21.3 \pm 2.0}$	0.009 ± 0.007	0.014 ± 0.012	$\mathbf{0.024 \pm 0.015}$	345.2 ± 155.8	$\mathbf{34.0 \pm 14.1}$
Pruning of both hidden and input neurons									
SCG	–	–	39.0 ± 3.8	22.3 ± 1.5	0.008 ± 0.005	0.013 ± 0.011	0.028 ± 0.015	493.1 ± 117.6	12.1 ± 2.4
SCGIR	10^{-6}	–	27.7 ± 7.9	21.0 ± 2.4	0.011 ± 0.004	0.015 ± 0.013	0.028 ± 0.015	533.6 ± 154.9	27.2 ± 5.8
SCGS	–	$2 \cdot 10^{-5}$	$\mathbf{22.8 \pm 7.8}$	21.3 ± 1.9	0.011 ± 0.005	0.010 ± 0.011	$\mathbf{0.026 \pm 0.018}$	609.8 ± 167.2	1389.8 ± 470.0
SCGS	10^{-6}	$2 \cdot 10^{-5}$	$\mathbf{23.7 \pm 11.9}$	21.5 ± 5.4	0.013 ± 0.009	0.015 ± 0.015	$\mathbf{0.027 \pm 0.018}$	620.6 ± 168.0	1272.6 ± 333.8
SCGS-ES	–	$-2 \cdot 10^{-6}$	31.9 ± 3.8	21.9 ± 1.8	0.013 ± 0.007	0.019 ± 0.012	0.035 ± 0.019	491.3 ± 127.7	956.8 ± 221.0
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	29.5 ± 6.9	21.3 ± 2.1	0.019 ± 0.021	0.024 ± 0.015	0.033 ± 0.017	473.8 ± 115.7	974.3 ± 246.3
SCGSA	–	10^{-4}	$\mathbf{23.7 \pm 4.1}$	21.6 ± 2.3	0.009 ± 0.005	0.022 ± 0.011	$\mathbf{0.022 \pm 0.016}$	956.1 ± 303.0	$\mathbf{26.4 \pm 6.4}$
SCGSA	10^{-6}	10^{-4}	25.9 ± 5.7	22.0 ± 2.2	0.010 ± 0.004	0.016 ± 0.014	$\mathbf{0.023 \pm 0.014}$	824.4 ± 337.1	$\mathbf{44.1 \pm 13.7}$

Table 5.10: Experiment 5.2.2 – The performance of the SCG, SCGIR, SCGS and SCGSA methods with and without pruning on the **BIN3** data set using the 18-12-12-6 network topology. The stated values correspond to the mean and standard deviation over 100 random network initializations.

method	c_F	c_G	*arch*	E_{tr}	E_t	$E(n_t)$	*imp*	c	c_n	MSE_t	$MSE(n_t)$	*imp*	S_t	epochs	t(s)
Without pruning															
SCG	–	–	18-12-12	99.0 ± 46.4	51.8 ± 6.8	51.9 ± 6.5	1.00	0	0	0.54 ± 0.12	0.55 ± 0.12	1.00	0.11 ± 0.02	70.5	0.4
SCGIR	10^{-4}	–	18-12-12	104.0 ± 45.9	52.8 ± 5.1	52.8 ± 4.6	0.98	0	0	0.55 ± 0.09	0.56 ± 0.09	0.98	0.10 ± 0.03	67.8	0.8
SCGS	–	$2 \cdot 10^{-4}$	18-12-12	69.1 ± 35.0	43.1 ± 7.4	43.0 ± 7.7	1.21	0	0	0.31 ± 0.07	0.31 ± 0.07	1.77	0.06 ± 0.02	109.7	3718.5
SCGS	10^{-4}	$2 \cdot 10^{-4}$	18-12-12	65.5 ± 39.4	42.4 ± 8.8	41.6 ± 8.8	1.25	0	0	$\mathbf{0.30 \pm 0.09}$	$\mathbf{0.30 \pm 0.09}$	**1.83**	0.06 ± 0.02	117.2	3927.1
SCGS	10^{-5}	$2 \cdot 10^{-4}$	18-12-12	61.1 ± 35.4	$\mathbf{40.8 \pm 8.8}$	$\mathbf{40.8 \pm 8.2}$	**1.27**	0	0	0.32 ± 0.08	0.31 ± 0.08	1.77	0.06 ± 0.02	117.0	3448.8
SCGSA	–	$5 \cdot 10^{-4}$	18-12-12	71.8 ± 38.2	47.6 ± 7.9	48.3 ± 7.2	1.07	0	0	0.48 ± 0.11	0.49 ± 0.10	1.14	0.09 ± 0.02	90.3	0.7
SCGSA	10^{-6}	$5 \cdot 10^{-4}$	18-12-12	72.8 ± 41.1	$\mathbf{46.6 \pm 10.4}$	$\mathbf{47.5 \pm 9.2}$	**1.09**	0	0	$\mathbf{0.45 \pm 0.12}$	$\mathbf{0.45 \pm 0.12}$	**1.22**	0.09 ± 0.03	95.6	1.5
Pruning of hidden neurons															
SCG	–	–	18-8.3-7.7	101.1 ± 45.0	46.4 ± 6.0	47.2 ± 5.1	1.0	0	0	0.46 ± 0.10	0.47 ± 0.10	1.0	0.08 ± 0.02	340.2	2.6
SCGIR	10^{-5}	–	18-8.0-8.1	108.7 ± 52.1	47.4 ± 6.4	47.7 ± 6.2	1.0	0	0	0.47 ± 0.10	0.48 ± 0.09	1.0	0.09 ± 0.03	346.7	4.4
SCGIR	10^{-4}	–	**18-6.6-6.2**	118.0 ± 39.7	47.0 ± 4.1	48.2 ± 3.4	1.0	0	0	0.44 ± 0.10	0.46 ± 0.10	1.0	0.08 ± 0.02	299.3	4.0
SCGS	–	10^{-4}	18-9.5-9.4	61.3 ± 37.4	$\mathbf{38.8 \pm 8.5}$	$\mathbf{38.9 \pm 9.3}$	**1.2**	0	0	$\mathbf{0.28 \pm 0.11}$	$\mathbf{0.28 \pm 0.11}$	**1.7**	0.06 ± 0.03	425.8	4000.7
SCGS	10^{-5}	$2 \cdot 10^{-4}$	18-10.2-10	67.6 ± 44.2	$\mathbf{38.0 \pm 7.9}$	$\mathbf{38.3 \pm 7.9}$	**1.2**	0	0	$\mathbf{0.29 \pm 0.07}$	$\mathbf{0.29 \pm 0.07}$	**1.7**	0.05 ± 0.02	421.6	6164.8
SCGS	10^{-5}	10^{-4}	18-9.9-9.4	66.7 ± 30.9	$\mathbf{37.8 \pm 9.3}$	$\mathbf{38.6 \pm 10.1}$	**1.2**	0	0	$\mathbf{0.27 \pm 0.12}$	$\mathbf{0.27 \pm 0.12}$	**1.8**	0.05 ± 0.02	466.6	4379.2
SCGSA	–	10^{-3}	18-9.5-8.6	75.8 ± 45.7	42.3 ± 6.7	43.9 ± 6.0	1.1	0	0	0.36 ± 0.09	0.38 ± 0.08	1.2	0.06 ± 0.02	395.4	3.7
SCGSA	10^{-4}	10^{-3}	18-8.9-8.5	73.6 ± 42.8	41.9 ± 6.1	42.8 ± 5.3	1.1	0	0	$\mathbf{0.35 \pm 0.09}$	$\mathbf{0.36 \pm 0.10}$	**1.3**	0.05 ± 0.02	423.7	5.8
Pruning of input neurons															
SCG	–	–	6.0-12-12	0.0 ± 0.0	0.0 ± 0.0	14.6 ± 4.8	1.0	10	0	0.001 ± 0.001	0.123 ± 0.069	1.0	0.22 ± 0.17	601.0	3.7
SCGIR	10^{-5}	–	6.0-12-12	0.0 ± 0.0	0.0 ± 0.0	16.2 ± 6.8	0.9	10	0	< 0.001	0.146 ± 0.086	0.8	0.23 ± 0.16	601.0	8.4
SCGIR	10^{-6}	–	6.0-12-12	0.0 ± 0.0	0.0 ± 0.0	13.4 ± 4.6	1.1	10	2	< 0.001	0.113 ± 0.071	1.1	0.19 ± 0.14	601.0	8.4
SCGS	–	10^{-5}	6.0-12-12	0.9 ± 1.4	0.3 ± 0.5	$\mathbf{0.7 \pm 0.7}$	**20.9**	9	9	0.004 ± 0.002	$\mathbf{0.006 \pm 0.003}$	**20.5**	0.05 ± 0.04	581.6	3392.7
SCGS	10^{-5}	10^{-5}	6.0-12-12	1.8 ± 2.5	0.6 ± 0.8	1.9 ± 2.1	7.7	8	4	0.005 ± 0.004	0.011 ± 0.008	11.2	0.06 ± 0.05	543.2	5080.3
SCGS	10^{-5}	$2 \cdot 10^{-6}$	6.0-12-12	0.0 ± 0.0	0.0 ± 0.0	0.9 ± 1.3	16.2	10	7	0.002 ± 0.001	0.009 ± 0.004	12.3	0.11 ± 0.11	564.0	6036.0
SCGSA	–	10^{-4}	6.0-12-12	0.6 ± 1.3	0.2 ± 0.4	$\mathbf{2.9 \pm 2.1}$	**5.0**	10	7	0.003 ± 0.003	$\mathbf{0.015 \pm 0.009}$	**8.0**	0.04 ± 0.02	499.1	4.6
SCGSA	10^{-6}	10^{-4}	6.0-12-12	0.9 ± 1.4	0.3 ± 0.5	$\mathbf{2.9 \pm 1.9}$	**5.0**	10	7	0.004 ± 0.003	$\mathbf{0.014 \pm 0.007}$	**8.6**	0.06 ± 0.04	501.7	8.4
Pruning of both hidden and input neurons															
SCG	–	–	7-9.2-8.3	22.2 ± 23.7	7.5 ± 7.9	26.7 ± 9.2	1.0	–	–	0.04 ± 0.04	0.29 ± 0.13	1.0	0.36 ± 0.29	601.0	4.5
SCGIR	10^{-4}	–	7.9-9.3-8.9	17.8 ± 21.8	7.1 ± 7.5	27.3 ± 5.7	1.0	–	–	0.04 ± 0.04	0.26 ± 0.11	1.1	0.23 ± 0.17	601.0	9.0
SCGIR	10^{-5}	–	7.4-9.3-8.9	11.3 ± 16.5	3.9 ± 5.6	24.6 ± 6.6	1.1	–	–	0.02 ± 0.03	0.29 ± 0.14	1.0	0.29 ± 0.26	601.0	8.9
SCGS	–	10^{-7}	6.8-9.2-8.4	19.5 ± 21.8	6.7 ± 7.2	13.2 ± 8.1	2.0	–	–	0.03 ± 0.03	$\mathbf{0.07 \pm 0.04}$	**4.1**	0.23 ± 0.12	601.0	3593.5
SCGS	10^{-5}	10^{-7}	7.4-9.3-8.9	14.4 ± 26.3	4.8 ± 8.8	14.2 ± 9.9	1.9	–	–	0.03 ± 0.04	0.08 ± 0.07	3.6	0.17 ± 0.16	601.0	3499.3
SCGS	10^{-5}	$2*10^{-8}$	7.6-9.8-9.3	7.6 ± 14.6	$\mathbf{2.4 \pm 4.9}$	$\mathbf{12.5 \pm 5.1}$	**2.1**	–	–	$\mathbf{0.02 \pm 0.02}$	0.08 ± 0.04	3.6	0.23 ± 0.19	601.0	5787.9
SCGSA	–	$2 \cdot 10^{-4}$	8.3-9.6-9.8	20.1 ± 16.0	9.9 ± 9.2	16.0 ± 7.6	1.7	–	–	0.06 ± 0.06	0.08 ± 0.06	3.8	0.05 ± 0.03	601.0	5.9
SCGSA	10^{-6}	$2 \cdot 10^{-4}$	7-9.2-9.0	39.4 ± 28.9	14.7 ± 10.7	20.2 ± 9.9	1.3	–	–	0.05 ± 0.03	0.07 ± 0.03	4.1	0.08 ± 0.06	601.0	9.8
SCGSA	10^{-6}	$5 \cdot 10^{-5}$	6.7-9.2-8.7	14.6 ± 11.8	$\mathbf{5.2 \pm 4.2}$	$\mathbf{13.7 \pm 7.7}$	**1.9**	–	–	$\mathbf{0.03 \pm 0.02}$	$\mathbf{0.05 \pm 0.02}$	**5.3**	0.11 ± 0.07	601.0	10.4

Table 5.11: Experiment 5.2.2 – The performance of the SCG, SCGIR, SCGS and SCGSA methods with and without pruning on the **WB** data set using the 35-15-15-5 network topology and 10-fold cross-validation.

method	c_F	c_G	$arch$	E_{tr}	E_t	$E(n_t)$	imp	MSE_t	$MSE(n_t)$	imp	S_t	epochs	t(s)
Without pruning													
SCG	–	–	35-15-15	0.006 ± 0.008	0.062 ± 0.020	0.060 ± 0.012	1.0	0.085 ± 0.020	0.087 ± 0.018	1.0	0.04 ± 0.01	90.8	1.4
SCGIR	10^{-5}	–	35-15-15	0.005 ± 0.007	0.061 ± 0.021	0.057 ± 0.020	1.1	0.082 ± 0.024	0.086 ± 0.023	1.0	0.04 ± 0.01	98.7	4.4
SCGIR	10^{-4}	–	35-15-15	0.007 ± 0.005	0.054 ± 0.020	0.052 ± 0.021	1.2	0.082 ± 0.022	0.084 ± 0.026	1.0	0.03 ± 0.01	86.3	3.9
SCGS	–	10^{-4}	35-15-15	0.009 ± 0.004	0.043 ± 0.017	0.048 ± 0.018	1.2	$\mathbf{0.058 \pm 0.018}$	$\mathbf{0.062 \pm 0.019}$	**1.4**	0.02 ± 0.01	127.3	62146.8
SCGS	10^{-5}	10^{-4}	35-15-15	0.008 ± 0.003	0.044 ± 0.015	$\mathbf{0.045 \pm 0.013}$	**1.3**	0.064 ± 0.015	0.065 ± 0.017	1.3	0.02 ± 0.01	118.6	43106.2
SCGSA	–	10^{-3}	35-15-15	0.006 ± 0.007	0.049 ± 0.022	0.051 ± 0.024	1.2	0.071 ± 0.023	0.074 ± 0.023	1.2	0.03 ± 0.01	89.3	**2.3**
SCGSA	10^{-4}	10^{-3}	35-15-15	0.008 ± 0.006	0.052 ± 0.021	$\mathbf{0.047 \pm 0.022}$	**1.3**	$\mathbf{0.070 \pm 0.025}$	$\mathbf{0.071 \pm 0.027}$	**1.2**	0.03 ± 0.01	120.9	**7.1**
Pruning of hidden neurons													
SCG	–	–	35-10.9-7.0	0.004 ± 0.006	0.052 ± 0.020	0.058 ± 0.015	1.0	0.084 ± 0.026	0.087 ± 0.027	1.0	0.04 ± 0.02	297.4	8.3
SCGIR	10^{-5}	–	35-10.2-6.6	0.008 ± 0.011	0.047 ± 0.020	0.053 ± 0.017	1.1	0.120 ± 0.119	0.124 ± 0.118	0.7	0.04 ± 0.02	301.8	15.8
SCGS	–	$5 \cdot 10^{-5}$	35-10.7-5.4	0.010 ± 0.007	0.035 ± 0.016	$\mathbf{0.041 \pm 0.017}$	**1.5**	$\mathbf{0.056 \pm 0.018}$	$\mathbf{0.058 \pm 0.018}$	**1.5**	0.02 ± 0.01	294.2	74067.0
SCGS	10^{-5}	$5 \cdot 10^{-5}$	35-9.3-4.8	0.173 ± 0.084	0.174 ± 0.068	0.043 ± 0.015	1.4	0.268 ± 0.100	0.062 ± 0.016	1.4	0.02 ± 0.01	233.1	16604.4
SCGSA	–	10^{-3}	35-12.5-7.7	0.007 ± 0.005	0.040 ± 0.019	0.052 ± 0.022	1.2	0.070 ± 0.022	0.073 ± 0.025	1.2	0.03 ± 0.01	298.4	**10.5**
SCGSA	10^{-5}	10^{-3}	35-12.5-8.8	0.003 ± 0.004	0.037 ± 0.017	$\mathbf{0.039 \pm 0.020}$	**1.6**	$\mathbf{0.060 \pm 0.018}$	$\mathbf{0.062 \pm 0.020}$	**1.4**	0.03 ± 0.01	295.4	**16.9**
Pruning of input neurons													
SCG	–	–	20.1-15-15	0.012 ± 0.011	0.032 ± 0.018	0.039 ± 0.020	1.5	0.055 ± 0.022	0.059 ± 0.022	1.5	0.05 ± 0.02	254.5	7.7
SCGIR	10^{-5}	–	19.9-15-15	0.014 ± 0.013	0.032 ± 0.019	0.037 ± 0.020	1.6	0.053 ± 0.024	0.057 ± 0.024	1.5	0.05 ± 0.02	255.5	20.2
SCGS	–	10^{-4}	27.8-15-15	0.019 ± 0.011	0.038 ± 0.017	0.044 ± 0.012	1.4	0.054 ± 0.020	0.056 ± 0.017	1.5	0.02 ± 0.01	240.1	70523.3
SCGS	–	10^{-4}	24-15-15	0.016 ± 0.007	0.032 ± 0.011	$\mathbf{0.035 \pm 0.012}$	**1.7**	$\mathbf{0.048 \pm 0.016}$	$\mathbf{0.051 \pm 0.018}$	**1.7**	0.03 ± 0.01	271.7	71625.7
SCGSA	–	$2 \cdot 10^{-4}$	20.4-15-15	0.014 ± 0.013	0.029 ± 0.017	0.034 ± 0.017	1.7	0.051 ± 0.022	0.053 ± 0.022	1.7	0.04 ± 0.02	253.0	**11.2**
SCGSA	10^{-5}	$2 \cdot 10^{-4}$	20.2-15-15	0.013 ± 0.011	0.030 ± 0.017	$\mathbf{0.034 \pm 0.019}$	**1.8**	$\mathbf{0.049 \pm 0.020}$	$\mathbf{0.051 \pm 0.020}$	**1.7**	0.04 ± 0.02	260.7	**22.2**
Pruning of both hidden and input neurons													
SCG	–	–	21-12.8-9.0	0.005 ± 0.004	0.032 ± 0.013	0.047 ± 0.019	1.3	0.055 ± 0.019	0.060 ± 0.023	1.4	0.05 ± 0.02	301.0	10.6
SCGIR	10^{-6}	–	19.9-12-8.6	0.007 ± 0.007	0.028 ± 0.015	0.039 ± 0.022	1.6	0.055 ± 0.020	0.061 ± 0.022	1.4	0.05 ± 0.03	301.0	19.9
SCGIR	10^{-5}	–	21.4-12.2-8.4	0.008 ± 0.008	0.029 ± 0.011	0.039 ± 0.018	1.6	0.062 ± 0.032	0.067 ± 0.033	1.3	0.05 ± 0.02	302.8	18.9
SCGS	–	10^{-5}	21.9-11.9-8.2	0.017 ± 0.018	0.026 ± 0.008	$\mathbf{0.034 \pm 0.014}$	**1.7**	0.049 ± 0.020	0.052 ± 0.021	1.7	0.03 ± 0.02	307.9	49915.2
SCGS	10^{-5}	10^{-5}	22.6-12.3-7.5	0.009 ± 0.007	0.028 ± 0.015	$\mathbf{0.034 \pm 0.017}$	**1.7**	$\mathbf{0.045 \pm 0.020}$	$\mathbf{0.048 \pm 0.022}$	**1.8**	0.03 ± 0.01	302.0	48910.1
SCGSA	–	10^{-3}	22.6-12.7-10.3	0.018 ± 0.010	0.027 ± 0.009	0.033 ± 0.019	1.8	$\mathbf{0.043 \pm 0.014}$	$\mathbf{0.045 \pm 0.017}$	**1.9**	0.04 ± 0.01	301.0	**13.2**
SCGSA	10^{-6}	10^{-3}	23.7-13.1-9.0	0.017 ± 0.011	0.028 ± 0.012	$\mathbf{0.032 \pm 0.016}$	**1.9**	0.046 ± 0.014	0.048 ± 0.017	1.8	0.03 ± 0.01	307.3	**20.6**

by the values of c in Table 5.10. In such a case, none of the trained networks has been able to provide correct outputs for all input patterns. Anyway, especially the exact method for sensitivity inhibition (SCGS) considerably improves generalization (1.8-times for the **BIN3** data set and 1.4-times for the **WB** data set). For the SCGSA method compared to SCG, the improvement of $MSE(n_t)$ is smaller (1.2-times for both the **BIN3** and **WB** data sets). For **BIN3**, sensitivity inhibition combined with enforced condensed internal representation yields even better generalization.

5.2.3 Experiment 5.2.3 – Results on weight decay

Experiment setting

In Experiment 5.2.3, we compared the SCGIR, SCGS and SCGSA methods with the standard training algorithms GD and SCG and with the GD, SCG and SCGIR methods enhanced by the weight decay regularization technique [119] (GDWD, SCGWD, SCGIRWD). See Subsection 3.3.6 on page 61 for a detailed description of weight decay. The main aim of this experiment was to mutually compare the weight decay and the new-proposed regularization techniques for analytical and approximative sensitivity control and their effect on the generalization abilities of BP-networks trained without pruning.

The tests involved two types of data: binary (*Binary Addition* and *Binary Multiplication* tasks – data sets **BIN2** and **BIN3**) and continuous (*World Bank* task – data set **WB**). In this experiment, the parameters $c_F \geq 0$ and $c_G \geq 0$ reflect the trade-off between the influence of the particular penalty terms in the overall error function. The parameter c_F corresponds to the representation error function (defined by Equation (4.1)). For the GDWD, SCGWD and SCGIRWD methods, the parameter c_G belongs to the weight decay error function (defined by Equation (3.59)). For SCGS and SCGSA, c_G corresponds to the sensitivity error functions defined by Equations (4.25) and (4.73), respectively.

All BP-networks were trained without pruning, the parameter $maxEpochs$ was set to 3000. The parameters c_F and c_G were set experimentally, separately for each task. For the **BIN2** and **BIN3** data sets, all the trained networks had the topologies 18-12-4 and 18-12-12-6, respectively. For the **WB** data set, we tested two topologies: 35-50-5 and 35-15-15-5. The results for the **WB** data set were obtained by a 10-times repeated 10-fold cross-validation. For all data sets, each tested method was applied to the same set of 100 different randomly initialized networks.

The results obtained for the **BIN2** and **BIN3** data sets are stated in Table 5.12. Table 5.13 summarizes the results for the **WB** data set. In the tables, the generalization abilities of the trained BP-networks and their sensitivity to noise in the data are indicated by the values of MSE_t, E_t, c and $MSE(n_t)$, $E(n_t)$, c_n, imp, respectively. We also compared the values of S_t and w_m.

Results – On the generalization abilities of the networks

The experiments proved, that training with weight decay improves the generalization abilities of BP-networks trained without pruning. For the **BIN3** data set, SCGWD outperforms in this respect both the SCGS and SCGSA methods,

Table 5.12: Experiment 5.2.3 – The performance of the SCG, SCGIR, SCGWD, SCGIRWD, GD, GDWD, SCGS and SCGSA methods (without pruning) on the **BIN2** data set (*Binary Addition* task) and on the **BIN3** data set (*Binary Multiplication* task). The stated values correspond to the mean and standard deviation over 100 random network initializations.

Binary Addition task – using the 18-12-4 network topology																
method	c_F	c_G	c	c_n	p_{IR}	$p_{IR,3}$	E_{tr}	E_t	$E(n_t)$	imp	MSE_t	$MSE(n_t)$	imp	S_t	w_m	epochs	t(s)
SCG	–	–	77	62	80	92	3.8 ± 15.5	3.9 ± 9.7	10.1 ± 12.5	1.0	0.054 ± 0.147	0.087 ± 0.146	1.0	0.07 ± 0.06	0.56 ± 0.08	415.2	1.5
SCGIR	10^{-4}	–	71	57	**88**	**97**	5.9 ± 20.0	4.9 ± 11.2	11.8 ± 13.6	0.9	0.065 ± 0.161	0.101 ± 0.157	0.9	0.07 ± 0.05	0.61 ± 0.11	998.3	7.7
SCGIR	10^{-6}	–	73	60	82	94	3.9 ± 15.6	3.9 ± 9.7	10.2 ± 12.6	1.0	0.054 ± 0.146	0.087 ± 0.146	1.0	0.07 ± 0.06	0.57 ± 0.09	428.2	3.4
SCGWD	–	-10^{-4}	75	53	61	88	3.2 ± 15.1	3.1 ± 9.3	9.0 ± 11.7	1.1	0.048 ± 0.145	0.082 ± 0.143	1.1	0.10 ± 0.03	0.38 ± 0.05	189.9	0.7
SCGIRWD	10^{-6}	-10^{-4}	76	50	61	88	3.2 ± 15.2	3.1 ± 9.3	9.0 ± 11.7	1.1	0.048 ± 0.145	0.083 ± 0.145	1.1	0.10 ± 0.04	0.38 ± 0.05	224.0	1.9
GD	–	–	18	17	52	81	22.3 ± 47.7	13.5 ± 19.3	20.1 ± 21.0	0.5	0.182 ± 0.282	0.212 ± 0.271	0.4	0.14 ± 0.04	0.34 ± 0.02	2691.1	11.7
GDWD	–	-10^{-3}	0	29	32	69	11.4 ± 36.8	10.6 ± 12.1	16.0 ± 12.6	0.6	0.107 ± 0.182	0.123 ± 0.180	0.7	0.14 ± 0.01	0.25 ± 0.02	1517.5	6.9
SCGS	–	$2 \cdot 10^{-5}$	59	84	81	93	1.3 ± 7.3	1.7 ± 5.6	$\mathbf{6.5 \pm 6.8}$	**1.6**	0.022 ± 0.068	$\mathbf{0.050 \pm 0.065}$	**1.7**	0.06 ± 0.03	0.53 ± 0.07	715.2	206.1
SCGS	10^{-6}	$2 \cdot 10^{-5}$	61	83	82	94	1.2 ± 7.1	1.6 ± 5.6	6.6 ± 6.8	1.5	0.021 ± 0.067	$\mathbf{0.050 \pm 0.064}$	**1.8**	0.06 ± 0.03	0.53 ± 0.06	764.7	224.5
SCGSA	–	0.0005	54	60	83	94	0.2 ± 1.1	1.0 ± 2.0	$\mathbf{5.8 \pm 3.1}$	**1.7**	0.012 ± 0.021	$\mathbf{0.048 \pm 0.024}$	**1.8**	0.06 ± 0.02	0.57 ± 0.07	708.4	3.8
SCGSA	10^{-6}	0.0005	52	69	81	94	2.6 ± 14.1	2.4 ± 7.6	7.6 ± 9.5	1.3	0.030 ± 0.089	0.065 ± 0.086	1.3	0.07 ± 0.04	0.55 ± 0.06	562.3	5.1
Binary Multiplication task – using the 18-12-12-6 network topology																	
method	c_F	c_G	c	c_n	p_{IR}	$p_{IR,3}$	E_{tr}	E_t	$E(n_t)$	imp	MSE_t	$MSE(n_t)$	imp	S_t	w_m	epochs	t(s)
SCG	–	–	0	0	62	84	128.8 ± 37.3	55.0 ± 4.5	55.2 ± 4.1	1.0	0.583 ± 0.084	0.594 ± 0.082	1.0	0.10 ± 0.02	0.45 ± 0.05	56.6	0.3
SCGIR	10^{-4}	–	0	0	63	85	130.0 ± 36.8	55.1 ± 4.7	55.3 ± 4.4	1.0	0.589 ± 0.078	0.601 ± 0.073	1.0	0.10 ± 0.02	0.46 ± 0.06	57.4	0.8
SCGIR	10^{-6}	–	0	0	62	84	129.4 ± 36.6	55.0 ± 4.5	55.2 ± 4.2	1.0	0.583 ± 0.083	0.594 ± 0.081	1.0	0.10 ± 0.02	0.45 ± 0.05	56.4	0.8
SCGWD	–	-10^{-3}	12	7	48	78	50.8 ± 63.7	33.5 ± 19.5	$\mathbf{35.4 \pm 17.8}$	**1.6**	0.269 ± 0.225	$\mathbf{0.281 \pm 0.223}$	**2.1**	0.07 ± 0.03	0.35 ± 0.04	218.8	1.1
SCGIRWD	10^{-6}	-10^{-3}	15	8	48	78	50.5 ± 63.8	33.5 ± 19.6	$\mathbf{35.3 \pm 17.8}$	**1.6**	0.268 ± 0.225	$\mathbf{0.281 \pm 0.222}$	**2.1**	0.07 ± 0.02	0.35 ± 0.04	210.0	2.8
GD	–	–	0	0	62	84	144.5 ± 25.8	57.7 ± 3.0	57.6 ± 3.0	1.0	0.630 ± 0.077	0.639 ± 0.074	0.9	0.11 ± 0.02	0.45 ± 0.05	225.6	1.5
GDWD	–	-10^{-3}	0	0	48	78	92.3 ± 50.8	51.0 ± 7.9	51.5 ± 7.5	1.1	0.428 ± 0.165	0.440 ± 0.163	1.3	0.09 ± 0.02	0.35 ± 0.04	569.4	3.9
SCGS	–	10^{-4}	0	0	**72**	**90**	74.9 ± 42.4	44.0 ± 10.1	44.3 ± 9.5	1.2	0.330 ± 0.121	0.342 ± 0.118	1.7	0.07 ± 0.02	0.51 ± 0.07	106.0	1704.6
SCGS	10^{-6}	10^{-4}	0	0	**71**	**90**	76.6 ± 42.1	44.5 ± 9.9	44.8 ± 9.6	1.2	0.330 ± 0.121	0.342 ± 0.119	1.7	0.07 ± 0.02	0.51 ± 0.07	103.9	1677.4
SCGSA	–	$5 \cdot 10^{-4}$	0	0	65	86	102.8 ± 45.9	52.3 ± 6.6	52.6 ± 6.0	1.1	0.519 ± 0.103	0.531 ± 0.099	1.1	0.10 ± 0.02	0.48 ± 0.06	73.1	0.7
SCGSA	10^{-6}	$5 \cdot 10^{-4}$	0	0	65	86	96.6 ± 49.4	51.8 ± 6.7	51.8 ± 6.5	1.1	0.504 ± 0.100	0.515 ± 0.096	1.2	0.10 ± 0.02	0.49 ± 0.07	77.9	1.3

Table 5.13: Experiment 5.2.3 – The performance of the SCG, SCGIR, SCGWD, SCGIRWD, GD, GDWD, SCGS and SCGSA methods (without pruning) on the **WB** data set (*World Bank* task). The stated values correspond to the mean and standard deviation over 100 random network initializations.

World Bank task – using the 35-50-5 network topology																
method	c_F	c_G	c	c_n	p_{IR}	$p_{IR,3}$	E_{tr}	E_t	$E(n_t)$	imp	MSE_t	$MSE(n_t)$	imp	S_t	w_m	epochs	t(s)
SCG	–	0	0	1	52	80	0.002 ± 0.004	0.041 ± 0.020	0.042 ± 0.020	1.0	0.102 ± 0.024	0.105 ± 0.024	1.0	0.09 ± 0.01	0.18 ± 0.02	130.6	3.0
SCGIR	10^{-4}	–	0	0	**78**	**92**	0.007 ± 0.007	0.051 ± 0.022	0.051 ± 0.021	0.8	0.115 ± 0.035	0.117 ± 0.035	0.9	0.07 ± 0.02	0.25 ± 0.08	120.8	8.6
SCGIR	10^{-6}	–	0	1	53	79	0.002 ± 0.004	0.041 ± 0.020	0.042 ± 0.020	1.0	0.103 ± 0.024	0.106 ± 0.023	1.0	0.08 ± 0.01	0.20 ± 0.06	129.0	9.7
SCGWD	–	-10^{-3}	0	3	***28***	***65***	0.002 ± 0.002	0.036 ± 0.018	0.037 ± 0.020	1.1	0.079 ± 0.017	$\mathbf{0.080 \pm 0.017}$	**1.3**	0.06 ± 0.01	0.13 ± 0.01	150.9	3.3
SCGIRWD	10^{-6}	-10^{-3}	0	3	***37***	***71***	0.002 ± 0.002	0.035 ± 0.019	0.036 ± 0.019	1.2	0.078 ± 0.017	$\mathbf{0.080 \pm 0.017}$	**1.3**	0.06 ± 0.01	0.14 ± 0.04	150.3	11.0
GD	–	0	0	1	53	79	0.015 ± 0.078	0.066 ± 0.076	0.068 ± 0.075	0.6	0.323 ± 1.744	0.324 ± 1.743	0.3	0.09 ± 0.01	0.20 ± 0.06	496.2	18.8
GDWD	–	-10^{-3}	0	1	37	71	0.018 ± 0.077	0.057 ± 0.075	0.058 ± 0.074	0.7	0.292 ± 1.747	0.293 ± 1.746	0.4	0.07 ± 0.01	0.14 ± 0.04	496.2	18.3
SCGS	–	$2 \cdot 10^{-5}$	0	2	56	82	0.003 ± 0.003	0.035 ± 0.018	0.036 ± 0.019	1.1	0.074 ± 0.020	$\mathbf{0.075 \pm 0.020}$	**1.4**	0.05 ± 0.01	0.17 ± 0.03	156.2	539.2
SCGS	10^{-6}	$2 \cdot 10^{-5}$	0	1	56	83	0.003 ± 0.003	0.035 ± 0.018	0.036 ± 0.018	1.2	0.074 ± 0.020	$\mathbf{0.075 \pm 0.020}$	**1.4**	0.05 ± 0.01	0.17 ± 0.03	150.6	514.1
SCGSA	–	$2 \cdot 10^{-4}$	0	3	54	81	0.002 ± 0.004	0.038 ± 0.020	0.039 ± 0.019	1.1	0.092 ± 0.022	0.094 ± 0.022	1.1	0.07 ± 0.01	0.18 ± 0.02	140.6	6.3
SCGSA	10^{-6}	$2 \cdot 10^{-4}$	0	2	54	82	0.002 ± 0.003	0.037 ± 0.020	0.037 ± 0.019	1.1	0.092 ± 0.022	0.094 ± 0.022	1.1	0.07 ± 0.01	0.18 ± 0.02	136.0	13.0
World Bank task – using the 35-15-15-5 network topology																	
method	c_F	c_G	c	c_n	p_{IR}	$p_{IR,3}$	E_{tr}	E_t	$E(n_t)$	imp	MSE_t	$MSE(n_t)$	imp	S_t	w_m	epochs	t(s)
SCG	–	0	0	0	70	89	0.006 ± 0.008	0.060 ± 0.025	0.061 ± 0.025	1.0	0.083 ± 0.027	0.084 ± 0.027	1.0	0.04 ± 0.01	0.30 ± 0.04	95.2	1.6
SCGIR	10^{-4}	–	0	0	**88**	**96**	0.009 ± 0.008	0.065 ± 0.025	0.063 ± 0.024	1.0	0.088 ± 0.027	0.089 ± 0.026	0.9	0.03 ± 0.01	0.34 ± 0.04	92.6	4.6
SCGIR	10^{-6}	–	0	0	79	92	0.006 ± 0.008	0.061 ± 0.024	0.061 ± 0.025	1.0	0.083 ± 0.027	0.084 ± 0.027	1.0	0.04 ± 0.01	0.22 ± 0.01	94.2	4.8
SCGWD	–	-10^{-3}	0	1	***50***	***80***	0.005 ± 0.007	0.054 ± 0.024	0.054 ± 0.024	1.1	0.075 ± 0.023	0.075 ± 0.023	1.1	0.04 ± 0.01	0.21 ± 0.01	99.5	1.7
SCGIRWD	10^{-6}	-10^{-3}	0	0	***37***	***74***	0.004 ± 0.006	0.054 ± 0.023	0.055 ± 0.023	1.1	0.074 ± 0.022	0.075 ± 0.022	1.1	0.04 ± 0.01	0.18 ± 0.01	99.8	4.9
GD	–	0	0	0	48	79	0.024 ± 0.021	0.067 ± 0.031	0.068 ± 0.032	0.9	0.105 ± 0.031	0.106 ± 0.031	0.8	0.05 ± 0.01	0.22 ± 0.01	283.0	7.3
GDWD	–	-10^{-3}	0	0	37	74	0.027 ± 0.018	0.066 ± 0.029	0.067 ± 0.029	0.9	0.106 ± 0.027	0.107 ± 0.027	0.8	0.05 ± 0.01	0.18 ± 0.01	288.4	7.1
SCGS	–	10^{-4}	0	10	**83**	**94**	0.009 ± 0.004	0.043 ± 0.017	$\mathbf{0.048 \pm 0.018}$	**1.3**	0.058 ± 0.018	$\mathbf{0.062 \pm 0.019}$	**1.3**	0.02 ± 0.01	0.34 ± 0.03	127.3	62147.0
SCGS	10^{-5}	10^{-4}	0	0	**84**	**95**	0.008 ± 0.003	0.044 ± 0.015	$\mathbf{0.045 \pm 0.013}$	**1.4**	0.064 ± 0.015	$\mathbf{0.065 \pm 0.017}$	**1.3**	0.02 ± 0.01	0.34 ± 0.03	118.6	43106.0
SCGSA	–	10^{-3}	0	0	**78**	**92**	0.010 ± 0.023	0.060 ± 0.032	0.058 ± 0.032	1.0	0.079 ± 0.038	0.079 ± 0.038	1.1	0.03 ± 0.01	0.33 ± 0.05	100.1	3.1
SCGSA	10^{-6}	10^{-3}	0	1	**79**	**92**	0.007 ± 0.007	0.053 ± 0.023	0.054 ± 0.025	1.1	0.074 ± 0.025	0.075 ± 0.025	1.1	0.03 ± 0.01	0.33 ± 0.04	94.8	6.0

while for the other data sets it achieves slightly worse results. Weight decay combined with enforced condensed internal representation (SCGIRWD) doesn't yield to further generalization improvements.

Contrary to SCGS and SCGSA, the methods based on weight decay remarkably reduce the absolute values of weights (w_m), while their effect on the overall network sensitivity S_t varies depending on the task being solved. When comparing the SCGWD method to SCG on the **BIN2** data set, the value of w_m has been reduced by 65%, while the overall network sensitivity raised by 43%. For the SCGS method compared to SCG, the value of w_m remains unchanged, while the overall network sensitivity has been reduced by 14%. These facts support the idea, that weight decay doesn't contribute to smoother network functions and it doesn't support sensitivity-based pruning as much as the sensitivity inhibiting techniques.

In sum, the experiments with BP-networks trained without pruning don't give a clear answer, whether the SCGWD-trained BP-networks generalize better or worse than the SCGS- and SCGSA-trained ones. However, they form different internal structures (as indicated by w_m and S_t). For SCGWD, the low values of weights together with relatively great sensitivities would influence the way in which the networks are pruned based on sensitivity analysis. In Experiment 5.5.2, we will prove, that for BP-networks trained together with pruning, the SCGS- and SCGSA-trained BP-networks remarkably outperform the SCGWD-trained ones in their generalization abilities.

5.2.4 Experiment 5.2.4 – Results on SCGIR

Experiment setting

In Experiment 5.2.4 [78], we concentrated on the SCGIR method with or without learning from hints and when compared to the GD, SCG and GDIR algorithms. We evaluated the prediction and generalization abilities of BP-networks trained by the respective methods. When training together with learning from hints (ALG-*hint*), we implemented the extra output hint method [106] (described in Subsection 3.4.2 on page 64).

The tests involved two tasks: *Binary Addition* (data set **BIN2A**) and *World Bank* (data set **WBA**). These data sets don't contain randomly generated input features. All methods were applied without pruning, the parameter *maxEpochs* was set to 6000.

For the **BIN2A** data set, we trained the networks with two topologies: 6-6-4 and 6-12-4. While 6-6-4 is the minimal network topology for the **BIN2A** data set, more than necessary hidden neurons are available in the second case. As the single hint output of the ALG-*hint* methods, we provided the carry-information to the second output bit. For the SCGIR and GDIR methods, the coefficient c_F was experimentally chosen 0.0005. For the GD and GDIR methods, the parameter α for the learning rates was set to 0.6. The whole **BIN2A** data set was used as the training set, while the networks were trained without early stopping. Each tested method was repeated 100-times on 100 different randomly initialized BP-networks.

For the **WBA** data set, the tested network topology was 25-37-1. In the experiments with hints, we used the Income group (*hintIG*) to create 5 hint

outputs that label the five classes of the Income Groups. The other option was to use clustering of the input data into k groups as a hint. For clustering, the c-means algorithm with Euclidean distance measure has been applied. We denote these variants as $hint9$ (for $k = 9$) and $hint6$ (for $k = 6$). For the GD and GDIR methods, the parameter α for the learning rates was set to 0.3. The coefficient c_F for GDIR was experimentally chosen 0.00004 and for SCGIR 0.0005. To compare the performance of tested methods, the 10-fold cross-validation has been used.

The results obtained for the **BIN2A** data set are stated in Table 5.14. Table 5.15 contains the results for the **WBA** data set. In the tables, we use the notation described in Table 5.4 and in Subsection 5.1.3 on page 122. In addition, c is the number of networks with no error on the training set, c_{IR} is the number of networks with a well-formed condensed internal representation (activities of all hidden neurons differ for all training patterns from the values $-0.85, 0, 0.85$ at most by 0.15). c_R is the number of networks with a well-formed condensed internal representation and no error on the training set. $n_s(0.15)$ represents the average number of hidden neurons with activities from one of the intervals $(-0.70, -0.15)$ or $(0.15, 0.7)$ (summed over all training patterns). The maximal possible value of $n_s(0.15)$ is for the **BIN2A** data set and the network topology 6-6-4 equal $64 \times 6 = 384$ and for the **WBA** data set $956 \times 37 = 35372$. $n_s(0.1)$ indicates the average number of hidden neurons with activities from one of the intervals $(-0.8, -0.1)$ or $(0.1, 0.8)$ (summed over all training patterns). n_{neur1} is the number of hidden neurons with an insignificant (small) weight to the output neuron, n_{neur2} indicates the average number of hidden neurons, that have formed always the same internal representation, or an identical or complementary one to another hidden neuron. Such neurons can be easily pruned from the network.

The prediction and generalization abilities of the trained BP-networks are indicated by the values of E_{tr}, c and in Table 5.14 and by the values of MSE_{tr} and MSE_t in Table 5.15.

Results – On the generalization abilities of the networks

For the **BIN2A** dataset, the prediction abilities of the trained BP-networks are about the same for all of the tested training algorithms, while the average error for SCG- and SCGIR-trained BP-networks is slightly lower than for their GD- and GDIR-trained counterparts. For networks with the topology 6-12-4, where more than necessary hidden neurons are available, all the methods have no difficulty to learn the task. On the contrary, only 25-45% of the networks with the minimal topology 6-6-4 have after training no error on the training set – depending on the chosen training algorithm. For the SCG- and SCGIR-trained networks with the topology 6-6-4, the presence of a hint often leads to a smaller error and to a higher number of adequately trained networks. For GD- and GDIR-trained networks, no such improvement is visible from the table.

For the **WBA** dataset, the SCGIR method reaches a better performance than the GDIR method – the final errors MSE_{tr} and MSE_t achieved for the training and test sets are apparently smaller. On the other hand, no significant generalization improvement is visible from Table 5.15 when training with a presence of a hint.

In sum, the SCGIR method outperforms in our tests the GDIR method in the prediction and generalization abilities of the trained BP-networks. Learning

Table 5.14: Experiment 5.2.4 – The performance of the SCGIR and related methods (without pruning) on the **BIN2A** data set (*Binary Addition* task) using the 6-6-4 and 6-12-4 network topologies. The stated values correspond to the mean and standard deviation over 10 random network initializations.

BIN2A data set – using the 6-6-4 network topology							
method	c	c_{IR}	c_R	n_{neur2}	$n_s(0.15)$	E_{tr}	epochs	t(s)
GD	37	3	6	–	21.94 ± 12.96	6.28 ± 7.03	4705.7	52.9
GD-*hint*	38	0	1	–	21.64 ± 11.71	5.98 ± 6.70	5046.3	53.0
GDIR	26	7	32	–	15.55 ± 15.43	7.90 ± 6.89	5001.0	90.6
GDIR-*hint*	26	11	44	–	10.98 ± 13.50	8.81 ± 7.44	5018.0	90.5
SCG	35	0	0	–	40.32 ± 20.49	5.91 ± 6.63	401.0	5.5
SCG-*hint*	45	1	2	–	39.68 ± 20.81	4.83 ± 6.15	403.0	5.6
SCGIR	37	6	11	–	25.32 ± 17.43	7.25 ± 8.29	400.6	9.0
SCGIR-*hint*	42	7	15	–	22.79 ± 16.60	5.34 ± 6.32	381.61	9.1
BIN2A data set – using the 6-12-4 network topology							
method	c	c_{IR}	c_R	n_{neur2}	$n_s(0.15)$	E_{tr}	epochs	t(s)
GD	99	0	0	0.20 ± 0.49	62.59 ± 28.23	0.02 ± 0.2	5001.0	59.2
GD-*hint*	100	0	0	0.34 ± 0.57	65.82 ± 22.31	0 ± 0.0	5002.0	58.5
GDIR	98	79	87	0.30 ± 0.48	0.80 ± 2.34	0.08 ± 0.6	5001.0	104.5
GDIR-*hint*	96	77	78	0.30 ± 0.56	0.98 ± 2.52	0.08 ± 0.4	5002.0	105.8
SCG	100	0	0	0.50 ± 0.67	58.04 ± 25.38	0 ± 0.0	497.1	6.2
SCG-*hint*	100	0	0	0.66 ± 0.82	62.35 ± 25.84	0 ± 0.0	492.6	6.3
SCGIR	100	30	30	0.37 ± 0.58	15.06 ± 16.54	0 ± 0.0	501.0	10.9
SCGIR-*hint*	100	50	50	0.59 ± 0.70	9.11 ± 16.30	0 ± 0.0	501.2	11.0

from hints has just marginal impact in this respect. This may be caused by the character of the hints, which are focused on the improvement of the transparency of the trained BP-networks rather than on better prediction and generalization.

5.2.5 Experiment 5.2.5 – Extended results on SCGIR

Experiment setting

In Experiment 5.2.5 [78], we investigated the behavior of the SCGIR method on noisy data. Here, we compare the GD, GDIR, SCG and SCGIR algorithms, all trained with the hint *hintIG* on the **WBA** data set. The experiment setting is analogical to Experiment 5.2.4 on page 137 with the following exceptions: The whole **WBA** data set was used as the training set. As the test set, we used the data derived from the original training set by adding $1 - 5$ % random noise to each of the input features. The tested network topology was 25-37-1. For GD and GDIR, the parameter α for the learning rates was set to 0.3. The coefficient c_F was experimentally chosen 0.00004 for GDIR and 0.0005 for SCGIR. The entire training process has been repeated 10-times for all the methods.

The results of Experiment 5.2.5 are summarized in Table 5.16. We compared the mean squared errors achieved for the training and (noisy) test sets (MSE_{tr}, $MSE(n_t)$ as well as the difference between internal representations of the original and of the noisy input data (mean over all activities of hidden neurons over all input patterns, *diff*).

Table 5.15: Experiment 5.2.4 – The performance of the SCGIR and related methods (without pruning) on the **WBA** data set (*World Bank* task) using the 25-37-1 network topology. The stated values correspond to the mean and standard deviation over 10 random network initializations.

method	epochs	t(s)	MSE_{tr}	MSE_v	MSE_t	n_{neur1}	n_{neur2}	$n_s(0.15)$	$n_s(0.1)$
GD	1709.1 ± 706.1	47.6 ± 0.8	0.046 ± 0.009	0.021 ± 0.003	0.060 ± 0.033	3.8 ± 1.9	4.0 ± 2.4	8992.5 ± 998.7	21133.0 ± 1879.0
*GD-hint*9	2498.4 ± 11.4	69.6 ± 0.4	0.050 ± 0.005	0.025 ± 0.003	0.068 ± 0.037	5.0 ± 1.4	3.5 ± 1.6	9040.8 ± 762.9	21217.0 ± 1214.8
*GD-hint*6	2430.5 ± 123.9	66.7 ± 0.9	0.051 ± 0.007	0.024 ± 0.002	0.067 ± 0.041	3.9 ± 1.2	5.3 ± 2.6	8170.9 ± 1252.2	19463.0 ± 2052.6
GD-hintIG	2480.3 ± 68.6	65.1 ± 0.7	0.047 ± 0.005	0.026 ± 0.004	0.066 ± 0.029	3.4 ± 1.5	3.6 ± 1.9	8477.6 ± 877.7	19900.0 ± 1632.5
GDIR	1182.2 ± 975.6	103.1 ± 2.4	0.076 ± 0.018	0.043 ± 0.010	0.107 ± 0.067	3.4 ± 1.3	9.0 ± 3.2	2776.9 ± 2092.3	9914.2 ± 5468.4
GDIR-*hint*9	2484.6 ± 54.7	195.3 ± 3.0	0.072 ± 0.009	0.043 ± 0.008	0.091 ± 0.048	4.7 ± 2.2	12.1 ± 2.3	1131.0 ± 228.8	4947.1 ± 814.7
GDIR-*hint*6	2441.8 ± 147.9	194.0 ± 1.8	0.070 ± 0.005	0.043 ± 0.008	0.092 ± 0.047	3.6 ± 1.5	9.0 ± 2.4	1405.4 ± 223.7	6001.9 ± 890.9
GDIR-*hintIG*	2453.2 ± 117.9	193.5 ± 0.5	0.076 ± 0.008	0.044 ± 0.007	0.090 ± 0.044	4.9 ± 1.9	8.6 ± 2.5	1330.9 ± 233.9	5610.0 ± 977.2
SCG	62.7 ± 11.6	5.0 ± 1.0	0.042 ± 0.009	0.028 ± 0.006	0.056 ± 0.026	3.5 ± 2.0	4.7 ± 1.5	8670.1 ± 1441.9	20324.0 ± 2232.6
*SCG-hint*9	251.4 ± 85.9	19.1 ± 2.6	0.034 ± 0.004	0.030 ± 0.006	0.064 ± 0.034	7.5 ± 3.1	5.4 ± 2.8	2767.5 ± 1196.4	7283.0 ± 2683.5
*SCG-hint*6	154.0 ± 53.2	14.2 ± 2.1	0.040 ± 0.006	0.030 ± 0.004	0.071 ± 0.047	5.9 ± 2.3	3.8 ± 3.0	4984.9 ± 1937.7	12874.0 ± 4199.0
SCG-hintIG	188.0 ± 13.2	18.2 ± 3.6	0.030 ± 0.006	0.027 ± 0.005	0.053 ± 0.025	14.0 ± 2.9	0.8 ± 1.0	3885.4 ± 871.5	9886.8 ± 1829.7
SCGIR	226.5 ± 127.1	17.2 ± 3.4	0.067 ± 0.005	0.033 ± 0.008	0.087 ± 0.045	4.0 ± 1.9	8.6 ± 1.6	3820.8 ± 583.5	13469.0 ± 1547.8
SCGIR-*hint*9	233.3 ± 62.9	53.8 ± 13.8	0.056 ± 0.012	0.043 ± 0.014	0.075 ± 0.024	8.9 ± 2.5	6.9 ± 2.2	729.6 ± 346.5	3329.6 ± 1298.9
SCGIR-*hint*6	238.2 ± 64.0	38.6 ± 7.5	0.050 ± 0.009	0.036 ± 0.008	0.071 ± 0.026	8.8 ± 2.8	8.6 ± 1.6	786.0 ± 252.5	3338.9 ± 994.9
SCGIR-*hintIG*	180.6 ± 51.7	46.5 ± 12.3	0.050 ± 0.010	0.037 ± 0.009	0.084 ± 0.040	10.4 ± 2.7	8.1 ± 2.8	689.2 ± 143.8	2696.9 ± 507.7

Table 5.16: Experiment 5.2.5 – The performance of the SCGIR and related methods (without pruning) on the **WBA** data set (*World Bank* task) using the 25-37-1 network topology and noisy data. The stated values correspond to the mean and standard deviation over 10 random network initializations.

method	MSE_{tr}	$MSE(n_t)$	*diff*	n_{neur1}	n_{neur2}	epochs	$n_s(0.15)$	$n_s(0.1)$
GD	0.0202	0.0233	0.0097 ± 0.0047	0.8	1.6	10002	7268.6	17147
GDIR	0.0427	0.0450	0.0056 ± 0.0082	6.8	4.3	10002	756.0	3712
SCG	0.0070	0.0132	0.0100 ± 0.0141	10.1	2.1	985.2	2142.9	5494.5
SCGIR	0.0070	0.0136	0.0056 ± 0.0165	16.8	3.9	1292.0	404.9	1632

Results – On the generalization abilities of the networks

In this test, the SCGIR method proved to perform better and to be less sensitive to the noise in the data than the GDIR method – the final errors MSE_{tr} and $MSE(n_t)$ achieved for the training and noisy test sets are apparently smaller, they are comparable to the SCG method.

5.2.6 Summary of Generalization

In sum, the SCGS and SCGSA methods outperform both the SCG and SCGIR algorithms in their generalization abilities and in their low sensitivity to noise in the data. The SCGIR method is in these respects comparable to the SCG algorithm, while it however outperforms the classical GDIR-method. If pruning of input (and hidden) neurons is allowed, the SCGS and SCGSA methods improve generalization similarly. Otherwise, SCGS achieves slightly better results in this respect. The SCGS and SCGSA methods are also outstandingly stable on data corrupted by varying amounts of noise.

Both methods for sensitivity inhibition (SCGS and SCGSA) outperform training with noise-corrupted data (SCG*) used traditionally to improve network's generalization. Moreover, contrary to the weight decay technique (SCGWD), they don't reduce the absolute values of network weights and contribute remarkably to smoother network functions. The actual behavior of the SCGS and SCGSA methods, however, seems to depend on the character of the processed data and on the topology of the trained network. Based on the experiments performed so far, a higher improvement might be achieved for binary / discrete data and for networks with more hidden layers.

5.3 Speed

In this section, we will concentrate on the following questions:

1. How fast are the new-proposed methods (SCGIR, SCGS and SGSA) when compared to related techniques? What are their convergence rates?
2. Are the methods stable with respect to the choice of initial parameters?

Analogically to the previous section, we compare the time requirements of the SCGIR, SCGS and SCGSA methods with the standard training algorithms (i.e., SCG, GD) and other related techniques. The time costs of the training process will be measured by the average elapsed training time in seconds (t(s)). Convergence rates of the respective training algorithms are indicated by the average number of training epochs (epochs).

The SCGS and SCGSA methods have two important optional parameters, c_F and c_G. They reflect the trade-off between the performance error function (defined by Equation 1.8), the representation error function (defined by Equation (4.1)) and the sensitivity error function (defined for SCGS by Equation (4.25) and for SCGSA by Equation (4.73)) in the overall error function. The SCGIR method uses just the parameter c_F. To evaluate the robustness of SCGIR, SCGS and SCGSA to the choice of parameters c_F and c_G, we evaluated the performance of the methods for various settings of these parameters.

In the following paragraphs, we will describe the settings and results of the experiments performed. The notation used in the experiments is described in Table 5.4 and in Subsection 5.1.3 on page 122.

5.3.1 Experiment 5.3.1 – General results

Experiment setting

In Experiment 5.3.1 evaluated partly in [81, 83, 84], we compared the time-costs of the SCGIR, SCGS and SCGSA methods with the standard SCG and GD training algorithms and other related techniques (SCG*, SCGIR*, GDWD, SCGWD, SCGIRWD). We used the settings of the experiments 5.2.1, 5.2.2 and 5.2.3 described on pages 123, 128 and 134, respectively. We considered BP-networks trained with and without pruning or with pruning of only hidden or only input neurons. The tests involved two types of data: binary (data sets **BIN2** and **BIN3**) and continuous (data set **WB**).

The results obtained for BP-networks trained with pruning are stated in Tables 5.5 and 5.6. Tables 5.7, 5.8, 5.9, 5.10 and 5.11 summarize the experiment for BP-networks trained without pruning and with pruning of only hidden or only input neurons. Tables 5.12 and 5.13 contain the results for the weight decay technique (algorithms GDWD, SCGWD, SCGIRWD) and BP-networks trained without pruning.

Results – On the time requirements of the methods

The experiments done on all three data sets show an interesting result: all of the tested SCG-based techniques (i.e., SCG, SCGIR, SCGS, SCGSA, SCG*,

SCGWD) have comparable convergence rates – they use during training similar number of epochs. On the other hand, the respective methods differ in the average training time indicated by the parameter t(s). The training time required by SCGIR and SCGSA is comparable to SCG and about 2-3 orders of magnitude lower than for SCGS depending on the network size. In all tests, SCGIR is slightly faster than SCGSA, while both methods are about 2-times slower than SCG, independently of the BP-network topology and of the training data. On the contrary, time costs of the SCGS method increase excessively with the growing number of hidden layers.

In our tests, the SCG* method (training with jitter) has been faster than SCGIR and SCGSA (as indicated by the values of t(s) in Tables 5.5 and 5.6). However, the training time of SCG* grows relatively fast with the size of the trained BP-network. For example: for the **WB** data set and the network topology with one hidden layer, SCG* is about 1.3-times slower than SCG, while for the network topology with two hidden layers, it is about 1.8-times slower. Based on this trend we expect, that the SCGIR and SCGSA methods might outperform SCG* in their speed for larger network topologies.

On the contrary to training with jitter (SCG*), the effect of weight decay (SCGWD, SCGIRWD) on the required training time was in most of the tests only marginal (as indicated by the values of t(s) in Tables 5.12 and 5.13).

5.3.2 Experiment 5.3.2 – Results on SCGIR

Experiment setting

In Experiment 5.3.2 [78], we concentrated on the SCGIR method and its time costs when enhanced with learning from hints [106]. We also compared the results to the GD, SCG and GDIR algorithms. We used the setting of Experiment 5.2.4 described on page 137. The tests involved two tasks: *Binary Addition* (data set **BIN2A**) and *World Bank* (data set **WBA**). All the BP-networks were trained without pruning.

The results obtained for the **BIN2A** data set are stated in Table 5.14. Table 5.15 contains the results for the **WBA** data set.

Results – On the time requirements of the methods

For both data sets, the SCGIR method proved to be much faster than the GDIR method. For the **BIN2A** dataset, the average number of training epochs is for SCGIR at least 10-times smaller than for GDIR, while for the **WBA** dataset, it is about 5-times smaller. Also the required training time in seconds is for SCGIR much lower than for GDIR (10-times for the **BIN2A** dataset and 6-times for the **WBA** dataset).

For the **BIN2A** data set, training together with learning from hints has only marginal effect on the convergence rates and training time of the methods. For the **WBA** data set, the training time of the SCGIR method increases due to learning from hints 2-3-times, while the average number of epochs remains about the same. The values of t(s) in the tables clearly show, that the required training time of the methods increases steadily with the growing number of hint outputs.

5.3.3 Experiment 5.3.3 – Stability test

Experiment setting

In Experiment 5.3.3 [81], we extended the setting of Experiment 5.2.1 by a test of the robustness of the SCGIR, SCGS and SCGSA methods to the choice of parameters c_F and c_G. The test was performed in two steps: first, the networks were trained with SCGS or SCGSA and an experimentally chosen value for $c_G = a * 10^b$. Afterwards, training was repeated with altered values of c_G with the exponent being randomly chosen from the interval $[b-1, b+1]$ with a uniform distribution. The same test was performed also for the parameter c_F and the SCGIR, SCGS and SCGSA methods.

The results obtained for the **BIN2** and **BIN3** data sets are summarized in Table 5.5. Table 5.6 contains the results for the **WB** data set.

Results – On the stability of the networks to the choice of initial parameters

The experiments done on all three data sets confirmed that both SCGS and SCGSA methods are quite stable with respect to the choice of the parameters c_F and c_G. For all the experiments, when changing the experimentally chosen value of c_G or c_F even 10-times, we were still able to achieve comparable results (as indicated, e.g., by the values of MSE_t, $MSE(n_t)$ and imp in Tables 5.5 and 5.6). Also the SCGIR method proved to be relatively stable to the choice of the parameter c_F, although it is apparently more sensitive to its choice than SCGS and SCGSA.

5.3.4 Summary of Speed

In sum, the training processes of the SCGIR, SCGS and SCGSA methods require similar number of epochs like the standard SCG training algorithm. They remarkably outperform in this respect the GD and GDIR methods. While the SCGS method suffers from high computational costs that grow excessively with the size of the network topology, the training times of SCGIR and SCGSA are much lower and appear to be proportional to the training time of SCG. The SCGIR and SCGSA methods require about twice the time spent by SCG to train a BP-network of an arbitrary topology. Both the SCGS and SCGSA methods also showed a very stable behavior with respect to the choice of the parameters c_F and c_G.

5.4 Transparency

In this section, we will summarize experimental results answering the following questions:

1. How powerful are the SCGIR, SCGS and SCGSA methods in forming the condensed internal representation of the trained BP-networks when compared to related techniques?
2. Are BP-networks trained by the respective methods likely to form a clear and transparent internal structure? Do the methods facilitate the following knowledge extraction?

Analogically to previous sections, we compared the SCGIR, SCGS and SCGSA methods with the standard SCG, GDIR and GD training algorithms and other related techniques that might influence the ability of a BP-network to form a condensed internal representation during training (e.g., ALG*, ALGWD and ALG-*hint*).

The notation used in the experiments is described in Table 5.4 and in Subsection 5.1.3 on page 122. The quality of the formed condensed internal representation is measured, e.g., by the values of p_{IR} and $p_{IR,3}$ (the percentages of the activities of hidden neurons that differ from the values -1, 0, and 1 at most by 0.1 and 0.3, respectively). In some of the experiments, we also evaluated the number of networks with a well-formed condensed internal representation and no error on the training, validation and test sets (c_R).

In a further set of experiments, we compared the structures and functions of concrete BP-networks trained by the SCG, SCGIR and SCGS training algorithms. These tests involved BP-networks with one and two hidden layers and data sets with a smaller number of input and output features (**BIN2A**, **BIN3A**). In the following paragraphs, we will describe the settings and results of the experiments performed.

5.4.1 Experiment 5.4.1 – General results

Experiment setting

In Experiment 5.4.1, we compared the SCGIR, SCGS and SCGSA methods with the standard SCG and GD training algorithms and other related techniques (SCG*, SCGIR*, GDWD, SCGWD, SCGIRWD). We investigated the abilities of the BP-networks trained by the respective methods to form a condensed internal representation during training.

Experiment 5.4.1 uses the settings of Experiments 5.2.1 and 5.2.3 described on pages 123 and 134, respectively. We considered BP-networks trained with and without pruning of hidden and input neurons. The tests involved two types of data: binary (*Binary Addition* and *Binary Multiplication* tasks – data sets **BIN2** and **BIN3**) and continuous (*World Bank* task – data set **WB**).

The results obtained for BP-networks trained with pruning of both hidden and input neurons are stated in Tables 5.5 and 5.6. Tables 5.12 and 5.13 summarize the experiment for BP-networks trained without pruning. In the tables, the quality of the formed condensed internal representation is indicated by the values

of p_{IR} and $p_{IR,3}$. In the experiments on the **BIN2** data set (in Table 5.5), we also compared the values of c_R.

Results – On the formed internal representation

The experiments done on all three datasets confirmed that all of the SCGIR, SCGS and SCGSA methods increase the chance that the BP-networks to form a condensed internal representation during training. The SCGIR method outperforms in this respect the SCGS and SCGSA methods. For SCGIR, the values of c_R and p_{IR} apparently increase with the growing value of parameter c_F. A strong enforcement of internal representation is, however, connected with worse generalization abilities of the trained BP-networks (indicated in the tables by the values of $MSE(n_t)$, $impr$). The SCGS and SCGSA methods seem to be in these respects less sensitive to the choice of parameter c_F.

When comparing the values of c_R in Table 5.5, the SCGIR method outperforms both the SCGS and SCGSA methods. It yields more BP-networks that learned the task and formed condensed internal representations during training. When comparing the SCGIR method to SCG, the value of c_R has raised 5-times, while for the SCGS and SCGSA methods compared to SCG, it has raised about 2-times. SCGS achieves slightly better results than the SCGSA method.

The experiments also proved that the SCGIR, SCGS and SCGSA methods remarkably improve the quality of the formed condensed internal representation indicated in the tables by the values of p_{IR} and $p_{IR,3}$. The highest increase of p_{IR} when compared to SCG was achieved for the **WB** data set and for BP-networks trained without pruning (this test is summarized in Table 5.6). In such a case, for BP-networks with two hidden layers, p_{IR} has raised by 26% for SCGIR, by 19% for SCGS and by 11% for SCGSA. For BP-networks trained with pruning (see Tables 5.5 and 5.6), the improvement of p_{IR} compared to SCG is smaller. The reason may be, that the hidden neurons that formed always the same representation, or an identical or complementary one to another hidden neuron, have been pruned from the networks.

The effect of training with jitter (SCG*, SCGIR*) on the formed internal representation was in the tests only marginal (as indicated by the values of c_R and p_{IR} in Tables 5.5, 5.6). On the other hand, the weight decay regularization technique (SCGWD, SCGIRWD) remarkably decreases the chance of the BP-network to form a condensed internal representation (as indicated by the values of p_{IR} and $p_{IR,3}$ in Tables 5.12 and 5.13). For example, for the **WB** data set and the network topology with one hidden layer, the value of p_{IR} is for SCGWD about 1.9-times smaller than for SCG. For the network topology with two hidden layers, it is about 1.4-times smaller. A combination of weight decay with learning internal representation (SCGIRWD) doesn't lead to great improvement of the formed internal representation.

5.4.2 Experiment 5.4.2 – Results on SCGIR

Experiment setting

In Experiment 5.4.2 [78], we concentrated on the SCGIR method and its ability to yield BP-networks with a well-formed condensed internal representation.

The SCGIR method was compared to the GD, SCG and GDIR algorithms. We also investigated the influence of learning from hints [106] on the quality of the formed condensed internal representation. We used the setting of Experiment 5.2.4 described on page 137. The tests involved two tasks: *Binary Addition* (data set **BIN2A**) and *World Bank* (data set **WBA**). All BP-networks were trained without pruning.

The results obtained for the **BIN2A** data set are stated in Table 5.14. Table 5.15 contains the results for the **WBA** data set. In the tables, the quality of the formed condensed internal representation is indicated by the values of $n_s(0.15)$ and $n_s(0.1)$. We also compared the number of networks with a well-formed condensed internal representation (indicated by c_{IR} and c_R) and the average number of hidden neurons that can be easily pruned from the network (n_{neur1}, n_{neur2}).

Results – On the formed internal representation

The experiments on the **BIN2A** and **WBA** data sets confirm the ability of the SCGIR method to form a condensed internal representation during training. On the other hand, the GDIR method seems to be even more powerful in this respect than SCGIR, especially for the **BIN2A** data set. However, the capability of SCGIR to form a transparent internal representation improves significantly when using simultaneously learning from hints.

For the **WBA** data set, the SCGIR method combined with learning from hints even outperforms the GDIR method (with and without learning from hints) – it yields a lower number of hidden neurons with non-transparent representations (indicated by the values of $n_s(0.15)$ and $n_s(0.1)$ in Table 5.15). Moreover, the SCGIR-*hintIG* method provides a higher value of $n_{neur1} + n_{neur2}$ than the other training algorithms. This means, that more hidden neurons can be pruned from the networks due to their output weights and representations.

5.4.3 Experiment 5.4.3 – Extended results on SCGIR

Experiment setting

In Experiment 5.4.3 [78], we used the setting of Experiment 5.2.5 described on page 139 and investigated the robustness of the formed networks' internal representations to noise in the data. Here, we compare the GD, GDIR, SCG and SCGIR methods, all trained with the hint *hintIG* on the **WBA** data set.

The results of the experiment are summarized in Table 5.16. The quality of the formed condensed internal representation is in the table indicated by the values of $n_s(0.15)$ and $n_s(0.1)$. We also evaluated the difference between internal representations of the original and of the noisy input patterns averaged over all activities of hidden neurons and over all input patterns (*diff*).

Results – On the formed internal representation

This experiment proved, that the SCGIR method has an outstanding ability to form a condensed internal representation during training. Moreover, the created network structures are very robust to noise in the input data. When considering the values of $n_s(0.15)$, $n_s(0.1)$ and *diff*, the SCGIR method in this experiment

outperforms all of the SCG, GD and GDIR methods. When comparing SCGIR to GDIR, the value of $n_s(0.15)$ is about two times smaller while the value of *diff* is about the same for both methods. The value of *diff* for the GDIR and SCGIR methods is half of the difference achieved for the GD and SCG methods.

5.4.4 Experiment 5.4.4 – Example network structures (for SCGIR-hint and SCG-hint)

Experiment setting

In Experiment 5.4.4 [78], we compared the function and internal structure of two concrete BP-networks trained by the SCGIR-*hint* and SCG-*hint* methods, respectively. Our goal was to assess, whether the BP-networks trained by SCGIR-*hint* are more likely to develop a transparent network structure when compared to their SCG-*hint*-trained counterparts. We also investigated, whether the formed network structures simplify knowledge extraction from the BP-network model.

The experiment setting is based on the setting of Experiment 5.2.4 described on page 137. The test involved the *Binary Addition* task (data set **BIN2A**). Both methods were applied without pruning to the same randomly initialized BP-network with the topology 6-6-4, the parameter *maxEpochs* was set to 6000, the coefficient c_F of the SCGIR method was chosen 0.0005. The whole **BIN2A** data set was used as the training set. The networks were trained without early stopping and without pruning. As the single hint output of both SCGIR-*hint* and SCG-*hint* methods, we provided the carry-information to the second output bit.

The characteristics of the two concrete BP-networks trained by SCGIR-*hint* and SCG-*hint*, respectively, are stated in Tables 5.17 and 5.18 and in Figure 5.3. Table 5.17 presents network activities of both BP-networks for a sample of 15 training patterns, while Table 5.18 contains the values of the absolute sensitivity coefficients averaged over all training patterns (in percents):

$$S_{uv} = 100 \operatorname*{mean}_p |S^p_{uv}|, \tag{5.3}$$

where p indexes all training patterns, v indexes all output neurons, and u indexes all network inputs. The sensitivity coefficients S^p_{uv} are defined by Equations (4.16) and (4.17). Figure 5.3 illustrates both the structures and the functions of the respective BP-networks.

Results – On the transparency of the formed internal structure

The values of activities of hidden neurons in Table 5.17 confirm, that the SCGIR-*hint* method has formed a transparent – nearly condensed – internal representation. For the SCG-*hint*-trained BP-network, the activities of hidden neurons are less transparent. Both BP-networks give correct outputs for all training patterns.

In Figure 5.3(a), we can clearly see that the BP-network trained by SCGIR-*hint* has succeeded in finding the actual computing algorithm. Moreover, the weights between its input and hidden layer are equal for the pairs of corresponding input neurons 1 and 4, 2 and 5, and 3 and 6. The first, third and fifth hidden neurons compute the 'carry' for higher output bits. The second, fourth and sixth

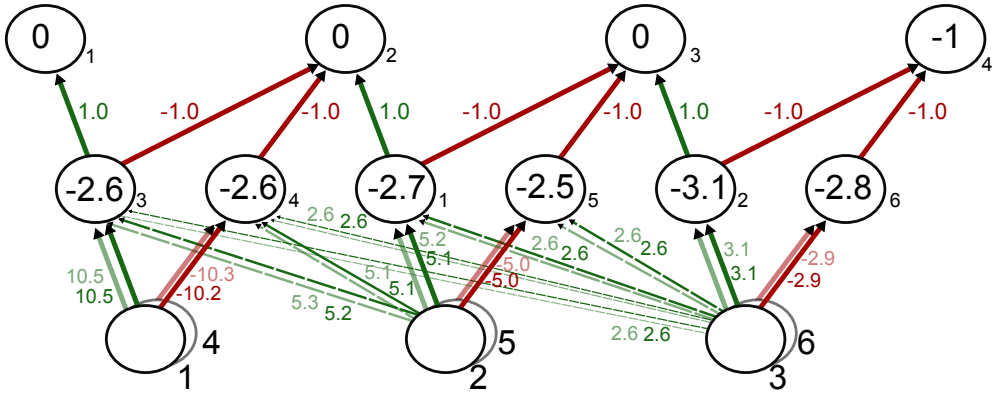
Table 5.17: Experiment 5.4.4 – Internal representation and network outputs of two BP-networks with the topology 6-6-4 trained by the SCGIR-*hint* and SCG-*hint* methods, respectively, on the **BIN2A** data set (*Binary Addition* task). The table comprises the results obtained for a sample of 15 training patterns.

Results for the BP-network trained by SCGIR-*hint*															
network inputs						network outputs				activities of hidden neurons					
−1	−1	−1	−1	−1	−1	−0.99	−0.99	−0.99	−0.99	−1.00	−1.00	−1.00	0.99	0.98	1.00
−1	−1	−1	−1	−1	1	−1.00	−1.00	−0.99	0.99	−1.00	−1.00	−1.00	1.00	1.00	−0.98
−1	−1	−1	−1	1	−1	−1.00	−1.00	0.99	−0.99	−1.00	−1.00	−1.00	1.00	−1.00	1.00
−1	−1	−1	−1	1	1	−1.00	−0.99	0.99	0.99	−0.99	−1.00	−1.00	1.00	−0.98	−0.99
−1	−1	−1	1	−1	−1	−1.00	1.00	−0.99	−0.99	−1.00	−1.00	−1.00	−1.00	0.98	1.00
−1	−1	−1	1	−1	1	−1.00	1.00	−0.99	0.99	−1.00	−0.99	−1.00	−1.00	1.00	−0.99
−1	−1	−1	1	1	−1	−1.00	1.00	0.99	−0.99	−1.00	−1.00	−1.00	−1.00	−1.00	1.00
−1	−1	−1	1	1	1	−1.00	0.99	0.99	0.99	−0.99	−1.00	−0.99	−0.99	−0.98	−0.99
−1	−1	1	−1	−1	−1	−1.00	−1.00	−0.99	0.99	−1.00	−1.00	−1.00	1.00	1.00	−0.98
−1	−1	1	−1	−1	1	−1.00	−1.00	0.99	−0.99	−1.00	1.00	−1.00	1.00	1.00	−1.00
−1	−1	1	−1	1	−1	−1.00	−0.99	0.99	0.99	−0.99	−1.00	−1.00	1.00	−0.98	−0.99
−1	−1	1	−1	1	1	−1.00	0.99	−0.99	−0.99	0.99	1.00	−1.00	1.00	0.99	−1.00
−1	−1	1	1	−1	−1	−1.00	1.00	−0.99	0.99	−1.00	−0.99	−1.00	−1.00	1.00	−0.99
−1	−1	1	1	−1	1	−1.00	1.00	0.99	−0.99	−1.00	1.00	−1.00	−1.00	1.00	−1.00
−1	−1	1	1	1	−1	−1.00	0.99	0.99	0.99	−0.99	−1.00	−0.99	−0.99	−0.98	−0.99
...						...				...					

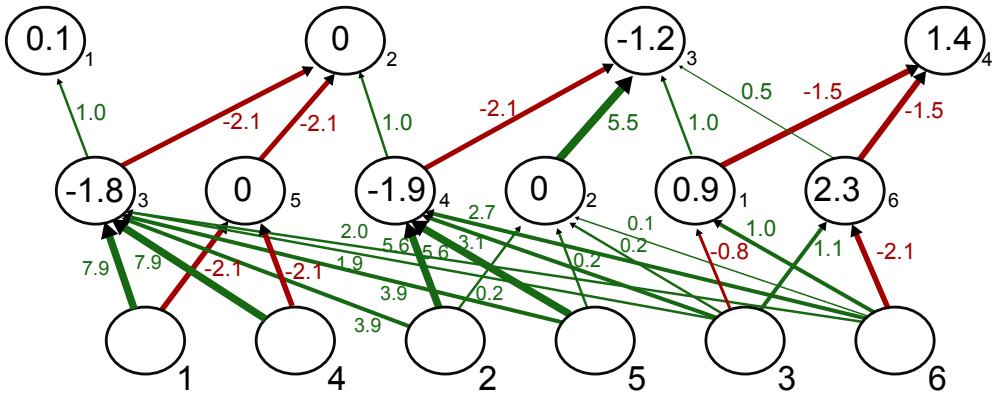
Results for the BP-network trained by SCG-*hint*															
network inputs						network outputs				activities of hidden neurons					
−1	−1	−1	−1	−1	−1	−1.00	−0.99	−0.96	−0.97	0.55	−0.56	−1.00	−1.00	1.00	1.00
−1	−1	−1	−1	−1	1	−1.01	−0.97	−1.00	0.99	0.99	−0.48	−1.00	−1.00	1.00	−0.75
−1	−1	−1	−1	1	−1	−1.02	−1.02	0.98	−1.00	0.60	−0.21	−1.00	−1.00	1.00	1.00
−1	−1	−1	−1	1	1	−1.03	−0.99	1.00	0.99	0.99	−0.12	−1.00	−0.98	1.00	−0.71
−1	−1	−1	1	−1	−1	−1.02	0.97	−0.96	−0.98	0.55	−0.56	−1.00	−1.00	0.06	1.00
−1	−1	−1	1	−1	1	−1.03	1.00	−1.00	0.99	0.99	−0.48	−1.00	−1.00	0.05	−0.75
−1	−1	−1	1	1	−1	−1.04	1.01	0.98	−1.00	0.61	−0.21	−1.00	−1.00	0.02	1.00
−1	−1	−1	1	1	1	−1.01	0.98	0.99	0.99	0.99	−0.11	−0.96	−0.98	0.01	−0.71
−1	−1	1	−1	−1	−1	−1.02	−1.01	−1.03	0.97	−0.77	−0.32	−1.00	−1.00	1.00	1.00
−1	−1	1	−1	−1	1	−1.02	−1.01	1.00	−0.98	0.74	−0.22	−1.00	−1.00	1.00	0.85
−1	−1	1	−1	1	−1	−1.04	−0.95	1.00	0.99	−0.73	0.08	−1.00	−0.91	1.00	1.00
−1	−1	1	−1	1	1	−1.02	1.00	−0.99	−0.99	0.77	0.18	−1.00	1.00	1.00	0.88
−1	−1	1	1	−1	−1	−1.04	0.98	−1.03	0.98	−0.77	−0.31	−1.00	−1.00	0.04	1.00
−1	−1	1	1	−1	1	−1.04	1.00	1.00	−0.98	0.74	−0.22	−1.00	−1.00	0.03	0.86
−1	−1	1	1	1	−1	−1.00	1.00	1.00	0.99	−0.73	0.09	−0.94	−0.91	0.00	1.00
...						...				...					

hidden neurons also compute similar functions for single output neurons. The BP-network trained by SCG-*hint* (depicted in Figure 5.3(b)) has also learned the 'carry' to the first and second output bits (the first and third hidden neuron) but the functions of other hidden and output neurons are less obvious.

The values of average sensitivity coefficients in Table 5.18 confirm the above-stated results. The sensitivity coefficients corresponding to the BP-network trained by SCGIR-*hint* are apparently regular. They are pairwise equal for the corresponding input neurons 1 and 4, 2 and 5, and 3 and 6. All of the output neurons are most sensitive to the values of the input features 3 and 6, while the pair of input features 2 and 4 is evaluated as the second most important. For the BP-network trained by SCGIR-*hint*, the sensitivity coefficients show less regularity. Moreover, their values for the pair of corresponding input neurons 3 and 6 are very different. The first and third output neurons are most sensitive to the values of the input features 2 and 4, which might be caused by the hint output, that provides the carry-information to the second output bit.



(a) The BP-network with the topology 6-6-4 trained by SCGIR-*hint*.



(b) The BP-network with the topology 6-6-4 trained by SCG-*hint*.

Figure 5.3: Experiment 5.4.4 – Network structures of two BP-networks with the topology 6-6-4 trained by the SCGIR-*hint* and SCG-*hint* methods, respectively, on the **BIN2A** data set (*Binary Addition* task). Positive weights are drawn green, negative weights red. The magnitudes of the weight values are illustrated by their thickness and indicated by numerical values, thresholds are shown inside the circles for the respective neurons. Redundant weights are not depicted.

5.4.5 Experiment 5.4.5 – Example network structures (for SCGS and SCG)

Experiment setting

In Experiment 5.4.5, we compared the functions and internal structures of two concrete BP-networks trained by the SCGS and SCG methods, respectively. Our goal was to evaluate, whether the SCGS-trained networks are more likely to develop a transparent network structure when compared to their SCG-trained counterparts. We also assessed, whether the formed network structures simplify knowledge extraction from the BP-network model.

The test involved the task of *Binary Multiplication of two 2-bit numbers* (data set **BIN3A**). Both methods were applied without pruning to the same randomly initialized BP-network with the topology 4-4-4-4, the parameter *maxEpochs* was set to 600, the coefficients c_F and c_G of the SCGS method were chosen 0.001 and 0.0001, respectively. The whole **BIN3A** data set was used as the training set, while the networks were trained without early stopping and without pruning.

Table 5.18: Experiment 5.4.4 – Average values of the absolute sensitivity coefficients (in percents) for two sample BP-networks with the topology 6-6-4 trained by the SCGIR-*hint* and SCG-*hint* methods, respectively, on the **BIN2A** data set (*Binary Addition* task). The coefficients S_{uv} are computed as $S_{uv} = 100\,\mathrm{mean}_p\,|S^p_{uv}|$, where p indexes all training patterns, v is an index over all output neurons and u is an index over all network inputs. The values of S_{uv} that are equal for the pairs of corresponding input neurons 1 and 4, 2 and 5, and 3 and 6, are colored red, green and blue, respectively.

Values of $S_{uv} = 100\,\mathrm{mean}_p\,\lvert S^p_{uv}\rvert$ for the BP-network trained by SCGIR-*hint*.							
S_{uv}		u (index of the input)					
		1	2	3	4	5	6
v (index of the output)	1	9 ± 12	13 ± 13	28 ± 18	9 ± 12	13 ± 13	28 ± 18
	2	5 ± 7	19 ± 12	26 ± 13	5 ± 7	19 ± 12	26 ± 13
	3	0 ± 0	11 ± 11	39 ± 11	0 ± 0	10 ± 11	39 ± 11
	4	1 ± 1	7 ± 8	43 ± 7	1 ± 1	5 ± 5	43 ± 7

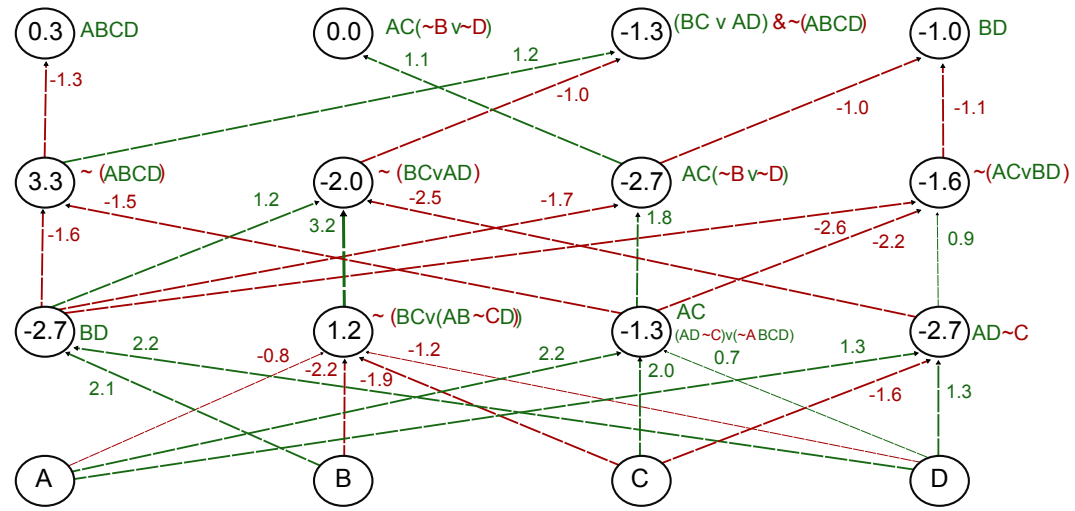
Values of $S_{uv} = 100\,\mathrm{mean}_p\,\lvert S^p_{uv}\rvert$ for the BP-network trained by SCG-*hint*.							
S_{uv}		u (index of the input)					
		1	2	3	4	5	6
v (index of the output)	1	18 ± 17	20 ± 11	15 ± 11	18 ± 17	20 ± 11	11 ± 13
	2	27 ± 20	14 ± 13	12 ± 11	27 ± 20	14 ± 13	6 ± 7
	3	1 ± 0	31 ± 3	18 ± 10	1 ± 0	31 ± 3	18 ± 8
	4	1 ± 1	1 ± 1	41 ± 6	1 ± 1	1 ± 1	56 ± 6

Both the structure and the function of the two concrete BP-networks trained by SCGS and SCG, respectively, is illustrated in Figure 5.4.

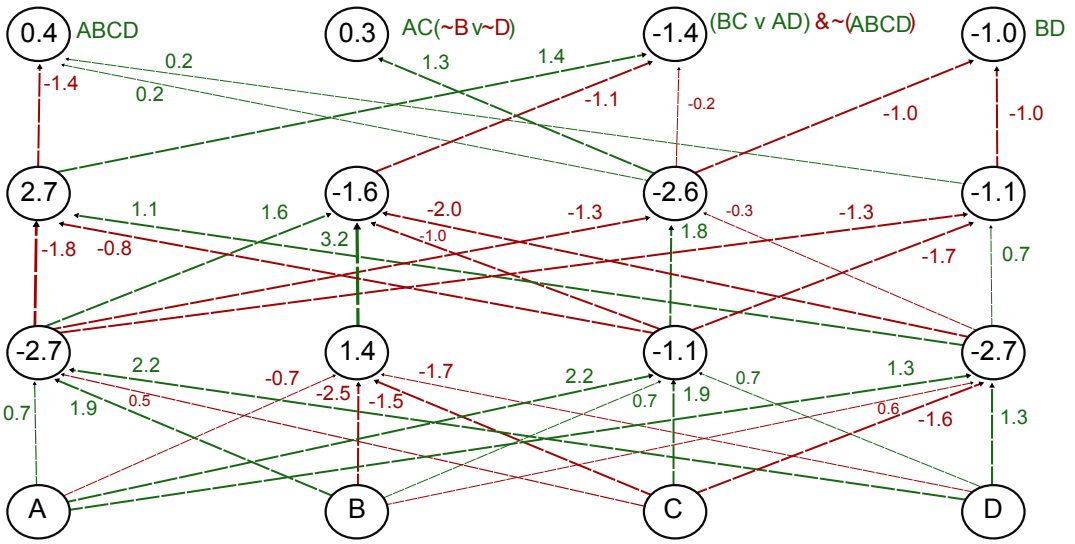
Results – On the transparency of the formed internal structure

Both BP-networks give correct outputs for all training patterns. The BP-network trained by the SCGS method (depicted in Figure 5.4(a)) clearly succeeded in finding a simple and transparent computational algorithm. Many weights are almost zeroed out, others are symmetrical, some of the weights between the input and hidden layer are equal for the pairs of corresponding input neurons A and C, B and D. The first hidden layer computes relatively simple functions over the input variables, while the second hidden layer computes slightly more complex functions over the outputs of the first hidden layer.

For the BP-network trained by the standard SCG method (depicted in Figure 5.4(b)), the functions of the hidden neurons are more complex and the network does not reveal such a clear computational pattern.



(a) The BP-network with the topology 4-4-4-4 trained by SCGS.



(b) The BP-network with the topology 4-4-4-4 trained by SCG.

Figure 5.4: Experiment 5.4.5 – Network structures of two BP-networks with the topology 4-4-4-4 trained by the SCGS and SCG methods, respectively, on the **BIN3A** data set (*Binary Multiplication* task). Positive weights are drawn green, negative weights red. The magnitudes of the weight values are illustrated by their thickness and indicated by numerical values, thresholds are shown inside the circles for the respective neurons. Redundant weights are not depicted. The attached expressions indicate the found functions denoted by means of *and*, *or*($\vee$) and *not*($\sim$).

5.4.6 Summary of Transparency

In sum, all of the SCGIR, SCGS and SCGSA methods significantly increase the chance that the BP-networks form condensed internal representation during training. When compared to SCG, they yield more transparent network structures that simplify knowledge extraction from the BP-network model.

The SCGIR method outperforms in these respects both the SCG and SCGS algorithms, it is however more sensitive to the choice of parameter c_F. Too large values of the coefficient c_F might namely result into BP-networks with a perfectly formed condensed internal representation yet incapable of approximating the desired function because of saturated hidden neuron outputs. The SCGIR method is less powerful in forming transparent internal representations when compared with GDIR, yet it yields comparable results for noisy data. The performance of the SCGIR method and its robustness to the choice of parameter c_F remarkably improves when combined with learning from hints or with the regularization techniques for exact or approximative sensitivity control (SCGS, SCGSA).

5.5 Structure optimization

In this section, we are interested in answering of the following questions:

1. How powerful and precise is the sensitivity-based relevance measure, when used to feature selection and pruning, in comparison to related techniques?
2. Is the developed training-and-pruning methodology together with the SCGIR, SCGS and SCGSA training algorithms likely to identify relevant input features and develop a (sub)optimal topology and a simple network structure during training?
3. Are the SCGIR, SCGS and SCGSA methods robust to the initial network topology and to the presence of redundant input features?
4. Do the created network structures enable an easy knowledge extraction from the model?

Most of the experiments in this section focus on the proposed training-and-pruning methodology based on internal representation and sensitivity analysis. It is described by Algorithms 4.3 and 4.4 on pages 89 and 90. We evaluated its qualities when combined with the SCG, SCGWD, SCGIR, SCGS and SCGSA training algorithms and when compared to alternative methods for pruning and feature-selection (e.g., pruning based on alternative relevance measures). In further experiments, we compared structures, functions and overall sensitivities of BP-networks trained by the respective training algorithms together with pruning.

The notation used in the experiments is described in Table 5.4 and in Subsection 5.1.3 on page 122. In the following paragraphs, we will describe the settings and results of the experiments performed.

5.5.1 Experiment 5.5.1 – Feature selection techniques

Experiment setting

The goal of Experiment 5.5.1 [83, 85] was to test the applicability of sensitivity- and cluster-based techniques to feature selection. We investigated, whether the tested feature selection techniques identify features relevant for the processed data and what relevance measures do best. Namely, we compared the distance-relevance [83] (*dist*), minimum-relevance [83] (*min*), maximum-relevance [83] (*max*), entropy-relevance [55] (*entro*) and sensitivity-relevance [34] (*sens*) measures. The general principle of the cluster-based feature selection techniques (*dist*, *min*, *max*, *entro*) is described in Subsection 3.2.1 on page 47. For a detailed description of the *sens* method, see Subsection 3.2.3 on page 52.

The test involved two tasks: *Binary Addition* (data set **BIN2**) and *World Bank* (data set **WB**). Both data sets contain a relatively high number of redundant, randomly generated input features. This enables us to evaluate the performance of the tested feature selection techniques. For the **BIN2** data set, the input features $[1, \ldots, 6]$ are necessary for the output, while the input features $[7, \ldots, 18]$ are randomly generated and thus irrelevant. For the **WB** data set, the input features $[1, \ldots, 25]$ are WDI indicators selected based on our domain knowledge – some of them might be more important for prediction than the others. The input features $[26, \ldots, 35]$ of the **WB** data set are randomly generated and surely redundant.

For all of the cluster-based feature selection techniques, we used the same clustering of the **BIN2** data set into 8 clusters and of the **WB** data set into 14 clusters. The clusters were generated by the c-means algorithm with Euclidean distance measure [23]. For the *sens* feature selection technique, we used sensitivity coefficients computed for concrete SCG-trained BP-networks with the topologies 18-12-4 (for the **BIN2** data set) and 35-50-5 (for the **WB** data set).

Table 5.19 shows the input features detected by the tested feature selection techniques as the most relevant.

Table 5.19: Experiment 5.5.1 – Input features selected by the respective methods from the **BIN2** data set (*Binary Addition* task). The last two columns indicate the total number of selected features and the total number of selected random features for the **WB** data set (*World Bank* task).

	Binary Addition task	*World Bank* task	
method	selected input features	total	random
sens	**[1 2 3 4 5 6]**	18	0
dist	[1 7 9 10 11 12 13 16]	17	0
min	[1 5 7 9 10 11 13 14 16]	18	1
max	[1 5 7 9 10 11 13 14 16 17 18]	18	1
entro	all	16	10

Results – On feature selection

Table 5.19 clearly shows that the sensitivity-based method (*sens*) outperforms all of the cluster-based feature selection techniques. For the **BIN2** data set, *sens* is the only method able to identify reliably all relevant input features $[1, 2, 3, 4, 5, 6]$. The other methods preferred irrelevant features instead. For the **WB** data set, only the *sens* and *dist* methods identified all of the randomly generated input features while they evaluated as relevant a reasonable number of the remaining input features (about 18). The indices of input features selected by the *sens* method for the **WB** data set were [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 20, 21, 22, 24].

5.5.2 Experiment 5.5.2 – Pruning techniques

Experiment setting

In Experiment 5.5.2, we concentrated on the new-proposed training-and-pruning methodology based on internal representation and sensitivity analysis. Namely, we assessed its ability to identify relevant input features and to develop BP-networks with a simple structure that generalize adequately. The methodology was combined with several training algorithms (SCG, SCGIR, SCGWD, SCGIR-WD, SCGSA). We evaluated, how the choice of the training algorithm affects the achieved results. In this experiment, we denote the training algorithm ALG combined with our training-and-pruning methodology as ALG-S.

For pruning of hidden and input neurons based on sensitivity analysis, our training-and-pruning methodology uses as the relevance measure the exact sensitivity coefficients S_{ij}^{p} suggested by Zurada et al. [126] and Fidalgo [34] (defined

by Equations (3.50) and (3.51)). We compare this approach (ALG-S) to the same pruning strategy, however with one of the alternative relevance measures – the so-called weight product [108] defined by Equation (3.48) (ALG-ST), consuming energy [43] defined by Equation (3.46) (ALG-CE), goodness factor [43] defined by Equation (3.45) (ALG-GF) and weight power [108] defined by Equation (3.47) (ALG-WP). See Subsection 3.3.3 for a detailed description of these approximative relevance measures.

The tests involved two types of data: binary (*Binary Addition* and *Binary Multiplication* tasks – data sets **BIN2** and **BIN3**) and continuous (*World Bank* task – data set **WB**). All BP-networks were trained together with pruning of both input and hidden neurons. The results for the **WB** data set were obtained by a 10-times repeated 10-fold cross-validation. Each tested method was applied to the same set of 100 different randomly initialized networks.

The setting of parameters is stated in Table 5.20. The parameter c_F corresponds to the representation error function (defined by Equation (4.1)). For the GDWD, SCGWD and SCGIRWD methods, the parameter c_G belongs to the weight decay error function (defined by Equation (3.59)). For SCGSA, c_G corresponds to the sensitivity error functions defined by Equations (4.25) and (4.73), respectively.

Table 5.20: Experiment 5.5.2 – Parameter setting.

data set	network topology	*maxEpochs*	method	c_F	c_G
BIN2	18-12-4	2001	SCGIR	10^{-6}	–
			SCGWD	–	10^{-3}
			SCGIRWD	10^{-6}	10^{-3}
			SCGSA	–	10^{-5}
			SCGIRSA	10^{-6}	10^{-5}
BIN3	18-12-12-6	1101	SCGIR	10^{-6}	–
			SCGWD	–	10^{-3}
			SCGIRWD	10^{-6}	10^{-3}
			SCGSA	–	$5 \cdot 10^{-5}$
			SCGIRSA	10^{-6}	$5 \cdot 10^{-5}$
WB	35-50-5	1201	SCGIR	10^{-6}	–
			SCGWD	–	10^{-3}
			SCGIRWD	10^{-6}	10^{-3}
			SCGSA	–	$2 \cdot 10^{-4}$
			SCGIRSA	10^{-6}	$2 \cdot 10^{-4}$
WB	35-15-15-5	301	SCGIR	10^{-6}	–
			SCGWD	–	10^{-3}
			SCGIRWD	10^{-6}	10^{-3}
			SCGSA	–	10^{-4}
			SCGIRSA	10^{-6}	10^{-4}

The results of the experiment are summarized in Tables 5.21 and 5.22. In the tables, we use the notation described in Table 5.4 and in Subsection 5.1.3 on page 122. In addition, H_1 and H_2 are the average numbers of neurons in the first and second hidden layers, respectively. The columns denoted by $BIN2$ and $BIN3$ comprise results for the **BIN2** and **BIN3** data sets, respectively. The columns denoted by $WB2$ and $WB3$ correspond to the **WB** data set and to BP-networks

with the initial topologies 35-50-5 and 35-15-15-5, respectively. Superior values of the tested criteria are indicated by darker color.

Results – On the ability to develop BP-networks with a simple and transparent structure that generalize well

The experiments done on all three datasets confirmed, that the exact sensitivity relevance measure (S) is more likely to correctly identify the important and redundant parts of the BP-network than its less precise yet computationally more efficient alternatives (ST, GF, CE, WP). Independently of the chosen training algorithm ALG, the generalization abilities of the ALG-S-trained and pruned BP-networks are apparently superior (as indicated in Table 5.21 by the values of MSE_t and $MSE(n_t)$).

The experiments don't give a clear answer, which of the tested relevance measures are able to prune higher numbers of hidden and input neurons than the other ones – different measures achieve better results than the others for different tasks, as indicated by the values of I, n_I, H_1 and H_2 in Table 5.21. All of the relevance measures have, however, a similar impact on the required training time and convergence rates (as indicated by the values of t(s) and epochs in Table 5.22).

When comparing the training algorithms, the weight decay-based methods (e.g., SCGWD-S, SCGWD-CE) seem to prune more input neurons than the other tested methods, but the generalization abilities of the pruned models are worse. In most of the experiments, the best-generalizing BP-networks were yielded by the SCGSA-S and SCGIRSA-S training algorithms.

5.5.3 Experiment 5.5.3 – General results

Experiment setting

In Experiment 5.5.3 [81, 83, 84], we investigated the robustness of the SCGIR, SCGS and SCGSA methods to the choice of the network topology and to redundant input features. We also evaluated the influence of the particular training algorithms on the ability of our training-and-pruning methodology to identify redundant input features and to adequately prune the BP-network during training. We compared the SCGIR, SCGS and SCGSA methods with the standard SCG training algorithm and other related techniques (SCG*, SCGIR*).

We used the settings of the experiments 5.2.1 and 5.2.2 described on pages 123 and 128, respectively. We considered BP-networks trained without pruning, with pruning of both input and hidden neurons, or with pruning of only hidden or only input neurons. The tests involved two types of data: binary (*Binary Addition* and *Binary Multiplication* tasks – data sets **BIN2** and **BIN3**) and continuous (*World Bank* task – data set **WB**).

The experiment setting of Experiment 5.2.2 on page 128 was extended in the following way: For the **WB** data set, we worked with three initial topologies: 35-50-5, 18-50-5 and 35-15-15-5. For the network topology 18-50-5, we worked with just 18 of the 35 input features that were selected by the *sens* feature selection technique in Experiment 5.5.1. Our goal was to compare the performance of the

Table 5.21: Experiment 5.5.2 – The performance of the listed methods (with pruning) on the **BIN2**, **BIN3** and **WB** data sets. The stated values correspond to the mean over 100 random network initializations. Superior values of the tested criteria are indicated by darker color. – Part I.

	MSE_t				$MSE(n_t)$				n_I				I				H_1				H_2	
	$BIN2$	$BIN3$	$WB2$	$WB3$	$BIN2$	$BIN3$	$WB2$	$WB3$	$BIN2$	$BIN3$	$WB2$	$WB3$	$BIN2$	$BIN3$	$WB2$	$WB3$	$BIN2$	$BIN3$	$WB2$	$WB3$	$BIN3$	$WB3$
SCG−S	0.003	0.036	0.060	0.052	0.035	0.399	0.064	0.054	82	42	93	82	7.8	7.5	22.3	21.0	7.4	9.5	35.3	11.7	8.8	7.5
SCGIR−S	0.003	0.037	0.058	0.055	0.036	0.342	0.063	0.058	84	42	96	80	7.6	7.5	22.3	21.3	7.5	9.5	35.0	11.6	8.9	7.7
SCGWD−S	0.008	0.015	0.055	0.058	0.033	0.035	0.057	0.059	99	88	92	86	6.0	6.5	22.3	20.6	8.6	9.7	43.1	11.6	9.6	7.5
SCGIRWD−S	0.009	0.016	0.056	0.058	0.033	0.035	0.058	0.060	100	90	87	84	6.0	6.4	22.9	21.0	8.6	9.7	43.4	11.7	9.5	7.4
SCGSA−S	0.002	0.046	0.051	0.055	0.028	0.077	0.054	0.058	90	40	95	87	6.9	7.7	22.1	20.6	7.5	9.7	32.2	11.5	9.1	7.1
SCGIRSA−S	0.003	0.048	0.051	0.055	0.029	0.080	0.054	0.057	84	40	95	86	7.4	7.5	22.4	21.2	7.8	9.5	31.5	11.7	8.9	7.7
SCG−ST	0.007	0.124	0.071	0.063	0.038	0.382	0.076	0.067	91	17	29	47	6.8	9.2	24.2	20.4	7.5	8.2	27.2	12.0	8.1	8.4
SCGIR−ST	0.009	0.120	0.072	0.065	0.040	0.386	0.076	0.068	88	20	23	46	7.0	9.0	25.4	20.7	7.5	8.3	28.6	11.7	8.2	8.1
SCGWD−ST	0.018	0.041	0.066	0.065	0.041	0.064	0.068	0.067	93	72	52	67	6.8	6.9	26.1	20.1	7.2	8.9	31.2	11.8	9.3	7.6
SCGIRWD−ST	0.018	0.031	0.066	0.065	0.041	0.052	0.068	0.066	93	73	52	64	6.8	6.9	25.8	20.8	7.2	9.1	32.4	12.3	9.4	8.5
SCGSA−ST	0.009	0.155	0.065	0.058	0.035	0.187	0.068	0.061	87	20	21	36	7.2	9.3	25.0	20.6	7.5	8.4	27.3	12.1	8.3	7.6
SCGIRSA−ST	0.009	0.127	0.065	0.065	0.035	0.159	0.068	0.068	89	15	16	45	7.0	9.0	26.0	20.5	7.4	8.3	27.7	12.0	8.4	8.0
SCG−GF	0.012	0.091	0.063	0.058	0.049	0.518	0.068	0.061	56	36	83	93	11.1	7.3	21.9	19.1	8.9	8.8	22.7	11.1	7.8	7.4
SCGIR−GF	0.012	0.087	0.061	0.064	0.049	0.467	0.067	0.067	58	39	85	93	10.8	7.2	21.2	19.0	8.7	8.8	22.3	11.4	7.8	7.6
SCGWD−GF	0.019	0.034	0.064	0.058	0.043	0.057	0.066	0.061	82	86	72	91	8.2	6.4	23.5	19.3	7.5	8.9	24.9	11.1	8.8	7.4
SCGIRWD−GF	0.019	0.033	0.064	0.056	0.043	0.056	0.067	0.058	82	89	70	90	8.2	6.2	23.9	19.6	7.5	8.8	25.3	10.7	8.6	7.3
SCGSA−GF	0.012	0.088	0.056	0.054	0.041	0.121	0.060	0.056	54	41	84	85	11.2	7.3	21.7	20.5	8.9	8.8	24.1	11.8	7.8	8.2
SCGIRSA−GF	0.013	0.097	0.057	0.054	0.042	0.130	0.060	0.056	56	33	78	90	11.1	7.7	22.7	19.3	8.9	9.0	25.4	11.5	8.0	7.5
SCG−CE	0.012	0.079	0.063	0.056	0.048	0.432	0.068	0.060	56	35	85	89	11.1	7.6	21.7	19.7	8.9	9.0	22.9	11.3	8.1	7.8
SCGIR−CE	0.010	0.074	0.062	0.059	0.045	0.457	0.067	0.063	57	38	85	91	11.0	7.6	21.4	19.2	8.8	9.0	21.5	11.5	8.2	7.9
SCGWD−CE	0.019	0.028	0.064	0.059	0.043	0.050	0.065	0.061	82	87	76	90	8.1	6.3	23.5	19.7	7.4	8.9	24.6	10.8	8.8	7.3
SCGIRWD−CE	0.019	0.027	0.064	0.060	0.043	0.050	0.066	0.062	83	89	75	90	8.0	6.2	23.3	19.6	7.4	8.9	23.7	10.8	8.7	7.4
SCGSA−CE	0.011	0.074	0.057	0.053	0.040	0.103	0.061	0.055	50	42	85	90	11.7	7.3	21.5	19.8	9.1	9.0	21.8	11.2	8.2	7.4
SCGIRSA−CE	0.011	0.075	0.056	0.056	0.042	0.106	0.060	0.057	51	42	84	90	11.6	7.2	21.6	19.8	9.0	8.7	22.9	11.9	8.1	7.8
SCG−WP	0.027	0.119	0.069	0.058	0.065	0.537	0.074	0.061	32	54	69	89	14.0	7.1	22.6	19.0	9.8	7.5	21.5	12.0	7.0	7.9
SCGIR−WP	0.028	0.187	0.068	0.060	0.067	0.574	0.073	0.064	31	58	68	86	14.0	6.8	22.7	19.8	9.8	7.4	22.1	12.1	6.9	8.2
SCGWD−WP	0.039	0.050	0.069	0.059	0.064	0.079	0.071	0.062	26	78	43	88	14.9	7.4	27.5	19.3	10.5	8.2	29.9	12.3	8.1	8.7
SCGIRWD−WP	0.039	0.049	0.069	0.056	0.064	0.077	0.071	0.059	26	76	40	90	14.9	7.7	28.0	19.1	10.5	8.3	30.0	12.7	8.3	9.0
SCGSA−WP	0.026	0.116	0.061	0.056	0.059	0.149	0.064	0.059	28	60	63	84	14.4	6.8	23.8	20.3	10.0	7.6	23.1	12.3	7.2	8.3
SCGIRSA−WP	0.024	0.107	0.062	0.058	0.055	0.140	0.067	0.060	35	56	56	89	13.6	6.8	25.1	19.3	9.7	7.6	24.6	12.2	7.2	8.0

Table 5.22: Experiment 5.5.2 – The performance of the listed methods (with pruning) on the **BIN2**, **BIN3** and **WB** data sets. The stated values correspond to the mean over 100 random network initializations. Superior values of the tested criteria are indicated by darker color. – Part II.

	S_t				w_m				p_{IR}				*epochs*				$t(s)$			
	BIN2	*BIN3*	*WB2*	*WB3*	*BIN2*	*BIN3*	*WB2*	*WB3*	*BIN2*	*BIN3*	*WB2*	*WB3*	*BIN2*	*BIN3*	*WB2*	*WB3*	*BIN2*	*BIN3*	*WB2*	*WB3*
SCG−S	0.09	0.32	0.11	0.05	1.36	1.20	0.29	0.51	88.9	77.2	54.9	73.4	1413.7	1001.0	680.8	305.1	5.3	6.4	16.7	10.8
SCGIR−S	0.08	0.29	0.10	0.05	1.41	1.19	0.29	0.52	93.0	77.3	56.7	74.4	1429.2	1001.0	688.5	304.6	10.0	12.5	41.3	18.1
SCGWD−S	0.33	0.14	0.09	0.06	0.59	0.53	0.16	0.51	42.1	48.8	27.2	73.4	378.4	1014.4	749.0	304.2	1.5	6.3	19.3	11.1
SCGIRWD−S	0.33	0.14	0.09	0.05	0.60	0.53	0.16	0.52	42.2	49.3	28.6	74.4	382.3	1008.2	767.8	302.1	2.9	12.5	51.1	19.3
SCGSA−S	0.09	0.10	0.08	0.04	1.35	0.90	0.32	0.52	90.4	72.5	60.7	74.5	1308.6	948.1	813.3	303.4	7.1	8.4	31.5	14.7
SCGIRSA−S	0.08	0.11	0.07	0.04	1.32	0.91	0.32	0.51	90.3	71.7	62.1	74.5	1378.3	960.3	810.6	305.2	11.8	13.4	55.7	23.4
SCG−ST	0.09	0.24	0.09	0.05	1.44	1.08	0.34	0.53	89.2	74.3	66.5	73.3	1413.9	997.4	1110.7	305.1	5.3	4.8	13.9	4.3
SCGIR−ST	0.09	0.31	0.09	0.05	1.45	1.08	0.31	0.52	93.0	74.3	66.1	73.1	1429.2	997.0	1100.5	304.6	10.0	10.2	31.8	12.2
SCGWD−ST	0.30	0.14	0.09	0.06	0.65	0.55	0.19	0.53	46.8	50.6	33.1	73.3	703.6	1007.7	1067.8	304.3	2.4	5.0	12.4	4.4
SCGIRWD−ST	0.30	0.14	0.09	0.05	0.64	0.54	0.19	0.52	46.9	50.4	33.2	73.1	701.7	1009.2	1081.6	302.1	4.0	10.9	36.8	12.6
SCGSA−ST	0.08	0.11	0.07	0.04	1.34	0.87	0.34	0.54	90.9	70.7	69.8	73.7	1239.7	927.0	1063.5	303.8	6.7	6.6	23.5	8.0
SCGIRSA−ST	0.08	0.10	0.07	0.05	1.37	0.87	0.33	0.52	91.9	71.2	69.3	73.1	1239.0	959.9	1064.0	305.0	10.4	11.6	46.8	16.1
SCG−GF	0.06	0.38	0.09	0.05	1.21	1.23	0.36	0.57	86.1	75.7	67.2	74.2	1416.2	1001.0	1044.4	304.9	5.2	5.1	12.8	4.3
SCGIR−GF	0.06	0.35	0.09	0.05	1.27	1.22	0.40	0.57	89.2	75.2	69.3	74.4	1429.2	1001.0	1065.3	304.6	9.8	10.5	32.3	12.0
SCGWD−GF	0.27	0.14	0.09	0.05	0.62	0.58	0.22	0.57	46.0	51.6	37.7	74.2	958.6	1008.7	905.6	303.0	3.2	5.1	9.7	4.2
SCGIRWD−GF	0.27	0.15	0.09	0.05	0.62	0.59	0.21	0.57	45.7	52.1	38.2	74.4	951.1	1005.3	899.2	300.8	5.5	10.8	26.4	11.8
SCGSA−GF	0.06	0.12	0.07	0.04	1.16	0.92	0.39	0.55	87.4	70.0	69.7	74.9	1222.7	953.7	1001.1	305.2	6.5	6.8	21.4	7.8
SCGIRSA−GF	0.06	0.11	0.07	0.05	1.14	0.91	0.37	0.59	88.0	71.0	69.7	75.3	1243.8	966.5	977.8	301.6	10.5	11.2	41.6	15.5
SCG−CE	0.07	0.32	0.09	0.05	1.23	1.21	0.38	0.57	86.4	76.3	67.3	74.1	1414.0	1001.0	1053.8	304.4	5.1	5.0	11.9	4.1
SCGIR−CE	0.06	0.29	0.09	0.05	1.25	1.20	0.39	0.57	89.5	76.2	70.4	74.2	1423.6	1001.0	1047.5	304.1	9.6	10.7	31.9	11.7
SCGWD−CE	0.27	0.15	0.09	0.05	0.62	0.58	0.21	0.57	46.1	51.4	37.6	74.1	953.1	1010.8	888.9	302.4	3.0	5.0	9.5	4.0
SCGIRWD−CE	0.27	0.15	0.09	0.05	0.62	0.58	0.22	0.57	45.9	51.7	39.3	74.2	952.1	1007.5	917.0	300.5	5.5	11.0	27.1	11.7
SCGSA−CE	0.06	0.11	0.07	0.04	1.12	0.93	0.41	0.58	87.7	71.6	72.6	75.3	1188.7	954.5	989.1	301.2	6.3	7.0	21.0	7.4
SCGIRSA−CE	0.06	0.11	0.07	0.05	1.12	0.96	0.39	0.55	88.6	71.6	72.1	74.2	1290.6	945.1	969.2	305.7	10.9	11.3	41.5	15.6
SCG−WP	0.06	0.43	0.09	0.06	1.06	1.36	0.42	0.58	85.1	75.8	69.6	74.0	1411.6	1001.0	821.0	305.1	5.1	5.0	9.5	4.3
SCGIR−WP	0.06	0.40	0.09	0.05	1.09	1.39	0.43	0.56	87.1	76.0	70.8	74.6	1426.6	1001.0	810.3	304.4	9.5	10.3	25.0	12.2
SCGWD−WP	0.17	0.15	0.08	0.06	0.38	0.62	0.20	0.58	40.6	53.4	37.6	74.0	709.3	1013.0	700.3	304.3	2.4	4.9	7.7	4.4
SCGIRWD−WP	0.17	0.15	0.08	0.06	0.38	0.61	0.19	0.56	40.4	53.3	37.5	74.6	739.7	1009.3	706.0	302.1	4.3	10.2	21.6	12.5
SCGSA−WP	0.05	0.13	0.07	0.05	0.99	0.98	0.43	0.57	85.4	71.2	72.8	74.8	1104.4	925.9	785.9	305.1	5.9	6.6	17.4	7.8
SCGIRSA−WP	0.05	0.13	0.07	0.05	1.09	1.00	0.40	0.55	87.4	71.8	71.9	74.4	1162.0	920.5	784.6	302.3	9.6	10.4	34.1	15.4

trained BP-networks, when the redundant input features are removed in advance and when they are pruned during training.

The results obtained for BP-networks trained with pruning of both hidden and input neurons are stated in Tables 5.5 and 5.6. Tables 5.7, 5.8, 5.9, 5.10 and 5.11 summarize the experiment for BP-networks trained without pruning and with pruning of only hidden or only input neurons. Table 5.23 contains the results for the **WB** data set and the network topology 18-50-5.

Results — On the ability to develop BP-networks with a simple and transparent structure that generalize well

All of the experiments proved the ability of our training-and-pruning methodology to correctly identify the important and redundant parts of the BP-network and to form a simple network structure during training (as indicated in the tables by the values of $arch$, I, H, n_I, n_H and n_A). For example, for the **BIN2** data set (in Table 5.5), the average final topology of the SCGS-trained BP-networks was 7-7.5-4, while about 87% of the networks pruned all of the redundant input features and about 21% of the networks finished training with the minimal topology for the *Binary Addition* task (6-6-4).

The effect of a concrete training algorithm (e.g., SCG, SCGIR, SCGS or SCGSA) on the average number of pruned hidden and input neurons varies depending on the task being solved and on the network topology. In our experiments with BP-networks with one hidden layer (in Tables 5.8 and 5.9), the SCGS and SCGSA methods pruned more hidden and input neurons than SCG. For BP-networks with two hidden layers (in Tables 5.10 and 5.11), all of the tested training algorithms yield similar topologies. Nevertheless, the choice of the training algorithm influences the internal structure and generalization abilities of the trained and pruned BP-networks (as we proved, e.g., in Experiments 5.2.1 and 5.2.2).

In all of our experiments, BP-networks trained together with pruning generalize better than BP-networks with the same initial topology trained without pruning. Pruning of redundant input features improves generalization more than pruning of only hidden neurons. For example, for the **BIN3** data set and the SCGSA-trained BP-networks (in Table 5.10), the value of $MSE(n_t)$ decreases about 9-times when pruning of both hidden and input neurons, 30-times when pruning of only input neurons and by 25%, when pruning of only hidden neurons.

Results – On the robustness of the methods to the network topology and redundant input features

The tests confirmed, that the presence of redundant input features makes the training process of BP-networks very difficult. In such a case, pruning is essential not only to improve generalization, but even to learn the task at all. This is mostly apparent for the tasks of *Binary Addition* (in Tables 5.7 and 5.8) and *Binary Multiplication* (in Table 5.10).

Moreover, BP-networks with initial topologies larger than necessary trained together with pruning outperform BP-networks with the optimal initial topologies and with less redundant input features.

For the task of *Binary Addition* (in Tables 5.7 and 5.8), only about 76% of the networks trained by SCG without pruning give a correct prediction. However,

Table 5.23: Experiment 5.5.3 - The performance of the SCG, SCGIR, SCGS and SCGS-ES methods with and without pruning on the **WB** data set using the 18-50-5 network topology and 10-fold cross-validation.

method	c_F	c_G	H	I	E_{tr}	E_v	E_t	epochs	t(s)
Without pruning									
SCG	–	–	50	18	0.013 ± 0.004	0.017 ± 0.008	0.034 ± 0.015	118.6 ± 23.2	2.1 ± 0.4
SCGIR	10^{-6}	–	50	18	0.016 ± 0.009	0.027 ± 0.010	0.032 ± 0.010	122.5 ± 40.5	8.9 ± 2.6
SCGS	–	$2 \cdot 10^{-5}$	50	18	0.014 ± 0.005	0.023 ± 0.009	$\mathbf{0.033 \pm 0.019}$	157.4 ± 80.0	242.4 ± 122.8
SCGS	10^{-6}	$2 \cdot 10^{-5}$	50	18	0.018 ± 0.009	0.023 ± 0.011	$\mathbf{0.032 \pm 0.012}$	126.5 ± 35.9	198.8 ± 97.5
SCGS-ES	–	$-2 \cdot 10^{-6}$	50	18	0.014 ± 0.004	0.019 ± 0.005	0.038 ± 0.018	113.9 ± 21.8	185.8 ± 36.3
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	50	18	0.020 ± 0.009	0.032 ± 0.007	0.033 ± 0.019	114.3 ± 25.1	226.8 ± 53.5
SCGSA	–	10^{-6}	50	18	0.011 ± 0.003	0.020 ± 0.018	$\mathbf{0.027 \pm 0.010}$	123.1 ± 19.0	5.0 ± 0.7
SCGSA	10^{-6}	10^{-6}	50	18	0.012 ± 0.003	0.022 ± 0.017	$\mathbf{0.027 \pm 0.010}$	125.3 ± 22.2	11.3 ± 2.0
Pruning hidden neurons									
SCG	–	–	36.9 ± 4.4	18	0.008 ± 0.003	0.032 ± 0.010	0.031 ± 0.018	320.5 ± 95.6	7.5 ± 2.1
SCGIR	10^{-6}	–	27.6 ± 11.0	18	0.011 ± 0.004	0.031 ± 0.011	0.033 ± 0.019	342.0 ± 73.6	19.4 ± 4.5
SCGS	–	$2 \cdot 10^{-5}$	30.3 ± 7.4	18	0.006 ± 0.003	0.022 ± 0.011	$\mathbf{0.029 \pm 0.013}$	535.6 ± 201.7	680.8 ± 236.0
SCGS	10^{-6}	$2 \cdot 10^{-5}$	25.2 ± 15.0	18	0.013 ± 0.004	0.016 ± 0.011	$\mathbf{0.031 \pm 0.009}$	497.3 ± 91.8	610.2 ± 109.5
SCGS-ES	–	$-2 \cdot 10^{-6}$	37.7 ± 5.8	18	0.010 ± 0.003	0.033 ± 0.015	0.031 ± 0.021	278.6 ± 80.1	544.5 ± 144.9
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	26.9 ± 9.3	18	0.010 ± 0.004	0.026 ± 0.014	0.032 ± 0.023	319.0 ± 59.5	625.6 ± 143.4
SCGSA	–	10^{-6}	26.1 ± 14.5	18	0.010 ± 0.005	0.023 ± 0.012	$\mathbf{0.029 \pm 0.018}$	637.1 ± 147.1	15.2 ± 2.3
SCGSA	10^{-6}	10^{-6}	26.8 ± 14.4	18	0.011 ± 0.005	0.023 ± 0.013	$\mathbf{0.031 \pm 0.020}$	611.4 ± 171.2	31.1 ± 8.4
Pruning of input neurons									
SCG	–	–	50	16.3 ± 1.6	0.018 ± 0.008	0.027 ± 0.017	0.034 ± 0.014	276.2 ± 70.2	9.8 ± 1.9
SCGIR	10^{-6}	–	50	15.5 ± 2.5	0.018 ± 0.008	0.033 ± 0.006	0.042 ± 0.020	329.5 ± 72.2	25.8 ± 5.9
SCGS	–	$2 \cdot 10^{-5}$	50	16.3 ± 2.4	0.019 ± 0.010	0.024 ± 0.011	$\mathbf{0.032 \pm 0.012}$	294.0 ± 72.1	405.8 ± 105.5
SCGS	10^{-6}	$2 \cdot 10^{-5}$	50	16.9 ± 1.3	0.013 ± 0.005	0.027 ± 0.013	$\mathbf{0.032 \pm 0.020}$	384.5 ± 86.6	544.4 ± 126.8
SCGS-ES	–	$-2 \cdot 10^{-6}$	50	15.9 ± 2.3	0.019 ± 0.008	0.030 ± 0.014	0.034 ± 0.017	322.4 ± 99.9	449.1 ± 107.0
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	50	15.9 ± 2.0	0.021 ± 0.009	0.028 ± 0.011	0.033 ± 0.014	298.0 ± 67.7	508.9 ± 69.8
SCGSA	–	10^{-6}	50	16.4 ± 1.3	0.015 ± 0.005	0.018 ± 0.014	$\mathbf{0.032 \pm 0.017}$	268.7 ± 137.7	13.5 ± 6.3
SCGSA	10^{-6}	10^{-6}	50	16.7 ± 1.6	0.011 ± 0.004	0.019 ± 0.015	0.034 ± 0.017	319.2 ± 169.4	29.8 ± 15.0
Pruning of both hidden and input neurons									
SCG	–	–	41.1 ± 4.6	18.0 ± 0	0.010 ± 0.003	0.027 ± 0.012	0.033 ± 0.016	1025.4 ± 117.5	13.4 ± 2.2
SCGIR	10^{-6}	–	30.3 ± 11.3	17.1 ± 1.912	0.013 ± 0.007	0.033 ± 0.013	0.037 ± 0.021	570.7 ± 130.8	27.4 ± 6.4
SCGS	–	$2 \cdot 10^{-5}$	35.7 ± 4.0	17.7 ± 1.0	0.009 ± 0.004	0.024 ± 0.014	$\mathbf{0.028 \pm 0.012}$	662.3 ± 133.3	821.2 ± 159.1
SCGS	10^{-6}	$2 \cdot 10^{-5}$	34.3 ± 7.7	18.0 ± 0	0.012 ± 0.003	0.022 ± 0.013	$\mathbf{0.032 \pm 0.017}$	577.4 ± 95.3	791.6 ± 108.1
SCGS-ES	–	$-2 \cdot 10^{-6}$	37.5 ± 6.2	16.4 ± 1.8	0.014 ± 0.007	0.032 ± 0.017	0.037 ± 0.022	471.2 ± 129.1	786.5 ± 201.8
SCGS-ES	10^{-6}	$-2 \cdot 10^{-6}$	32.5 ± 8.7	17.5 ± 1.6	0.012 ± 0.007	0.031 ± 0.015	0.032 ± 0.014	572.3 ± 164.9	842.6 ± 287.7
SCGSA	–	10^{-6}	37.3 ± 9.2	17.6 ± 1.3	0.011 ± 0.007	0.025 ± 0.012	$\mathbf{0.030 \pm 0.019}$	1017.9 ± 158.1	23.7 ± 5.9
SCGSA	10^{-6}	10^{-6}	35.8 ± 10.6	16.9 ± 2.4	0.013 ± 0.010	0.025 ± 0.015	0.033 ± 0.019	1005.1 ± 211.6	44.7 ± 8.8

both the SCGS and SCGSA methods remarkably increase the number of BP-networks that learned the task (indicated by the value of c). If pruning of inputs is allowed, almost all BP-networks learn the task correctly, independently of the chosen training algorithm. In comparison, networks trained with an optimum topology (6-6-4) and without redundant input features from scratch require a relatively high number of epochs (about five times as many) to converge while maintaining a much lower performance — only about 55-65% of the networks give a correct prediction.

For the *World Bank* task, we compared the performance of BP-networks with the initial topologies 35-50-5 (in Table 5.9) and 18-50-5 (in Table 5.23). Not surprisingly, when training without pruning, BP-networks with the topology 18-50-5 generalize better (as indicated by the value of E_t). On the other hand, if pruning of input (and hidden) neurons is allowed, BP-networks with the initial topology 35-50-5 achieve even better results, especially when trained by the SCGS and SCGSA methods.

5.5.4 Experiment 5.5.4 – Results on SCGIR

Experiment setting

In Experiment 5.5.4 [78], we concentrated on the SCGIR method and its ability to prune hidden neurons based on internal representation and sensitivity analysis. The SCGIR method was compared to the GD, SCG and GDIR training algorithms. We also investigated the influence of learning from hints [106] on the formed network structures. We used the setting of Experiment 5.2.4 described on page 137. The test involved the *World Bank* task (data set **WBA**). All BP-networks were trained without pruning.

Results are stated in Table 5.15. We compared the numbers of hidden neurons that could be pruned from the networks due to their representation (n_{neur2}) and because they have small absolute values of weights to the output neuron (n_{neur1}).

Results — On the ability to identify redundant hidden neurons

The experiments confirm the ability of the SCGIR method to identify redundant hidden neurons, especially when applied together with learning from hints. The SCGIR-hintIG method remarkably outperforms in this respect all of the other tested methods (with and without learning from hints).

When training without hints, the value of n_{neur2} is for SCGIR similar to GDIR and about two times greater than for GD and SCG. That means that SCGIR and GDIR can prune more hidden neurons than SCG and GD – most probably due to more transparent internal representations formed.

5.5.5 Experiment 5.5.5 – Example network structures (for SCGSA and SCG)

Experiment setting

In Experiment 5.5.5, we compared final topologies, functions and internal structure of two concrete BP-networks trained by the SCGSA and SCG methods, re-

spectively. Our goal was to assess, whether the BP-networks trained by SCGSA are more likely to develop a small, simple and transparent network structure when compared to their SCG-trained counterparts. We also investigated, whether the formed network structures simplify knowledge extraction from the BP-network model.

The experiment setting is based on the setting of Experiment 5.2.1 described on page 123. The test involved the *Binary Addition* task (data set **BIN2**). Both methods were applied with pruning of both input and hidden neurons to randomly initialized BP-networks with the topologies 18-12-4, the parameter $maxEpochs$ was set to 2001, the coefficients c_F and c_G of the SCGSA method were chosen as 0.0005 and 0.0005, respectively.

Figure 5.5 illustrates both the structures and the functions of the respective BP-networks (before pruning, and after pruning and retraining).

Results – On the ability of the methods to prune the network adequately

Figure 5.5 clearly shows that while both methods identified the 12 irrelevant input features, only the SCGSA method has pruned the network topology in an optimal way.

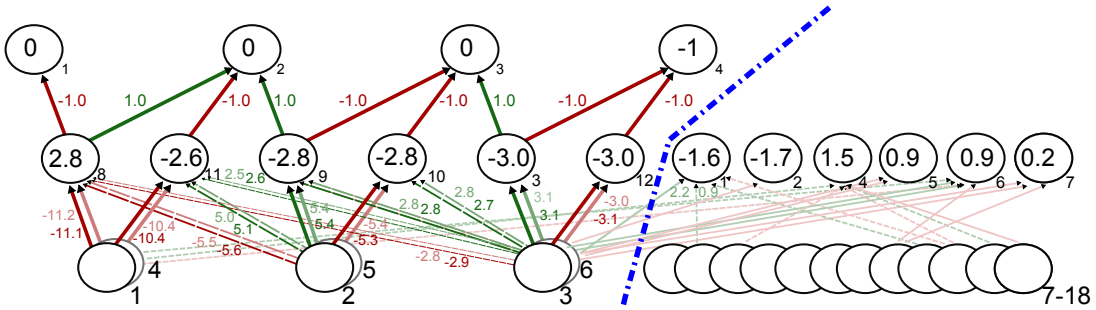
For the SCGSA-trained BP-network before pruning (showed in Figure 5.5(a)), the 6 hidden neurons on the right are redundant (with almost zero edges to all of the outputs) and can be pruned immediately. The 12 input neurons on the right are also redundant (as there are nearly zero edges from these inputs to the relevant hidden neurons) and can be pruned next. The remaining neurons form an optimal topology that solves the given task of *Binary Addition* (6-6-4, depicted in Figure 5.5(b)).

For the SCG-trained BP-network before pruning (showed in Figure 5.5(c)), the difference in the sensitivity coefficients between relevant and redundant input or hidden neurons is not so clear and pruning is thus more difficult. The final topology is therefore greater than necessary (6-8-4, depicted in Figure 5.5(d)). The SCG-trained BP-network also contains apparently more non-zero weights than the SCGSA-trained BP-network.

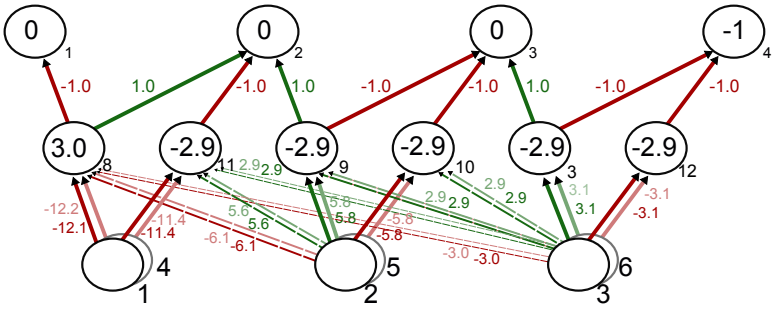
Results – On the transparency of the formed internal structure

Both SCGSA- and SCG-trained BP-networks give correct outputs for all training patterns. However, only the SCGSA-trained BP-network has a simple and clear internal structure (depicted in Figure 5.5(b)). It is similar to the structure of the BP-network trained by the SCGIR-hint method on the **BIN2A** data set in Experiment 5.4.4 (showed in Figure 5.3(a)).

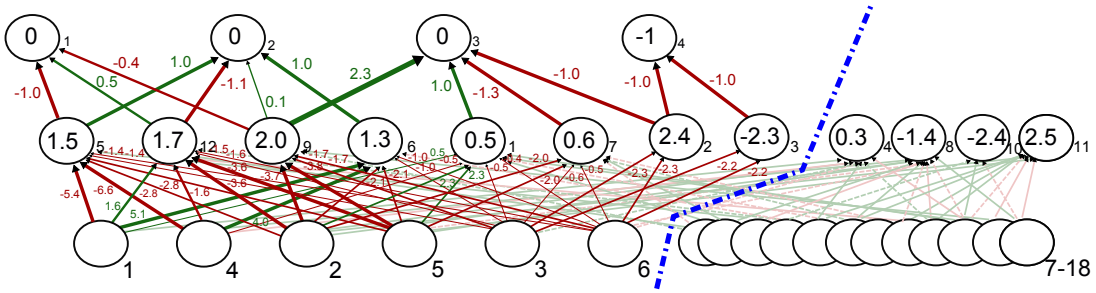
Many weights of the SCGSA-trained BP-network are zeroed out, the remaining weights between the input and hidden layer are equal for the couples of corresponding input neurons 1 and 4, 2 and 5, and 3 and 6. The first, third and fifth hidden neurons from the left (labeled by 8, 9 and 3, respectively) compute the 'carry' for higher output bits. The second, fourth and sixth hidden neurons (labeled by 11, 10 and 12, respectively) also compute similar functions for single output neurons.



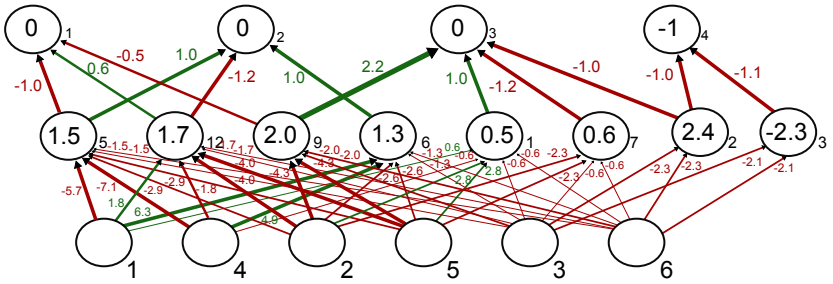
(a) The BP-network with the topology 18-12-4 trained by SCGSA – before pruning



(b) The BP-network with the topology 18-12-4 trained by SCGSA – after pruning and retraining.



(c) The BP-network with the topology 18-12-4 trained by SCG– before pruning.



(d) The BP-network with the topology 18-12-4 trained by SCG – after pruning and retraining.

Figure 5.5: Experiment 5.5.5 – Network structures of two BP-networks with the initial topologies 18-12-4 trained by the SCGSA and SCG methods, respectively, on the **BIN2** data set (*Binary Addition* task). Positive weights are drawn green, negative weights red. The magnitudes of the weight values are illustrated by their thickness and indicated by numerical values, thresholds are shown inside the circles for the respective neurons. Redundant weights are not depicted. The blue dash-dotted lines separate input and hidden neurons to be pruned from the networks.

The structure of the BP-network trained by SCG (depicted in Figure 5.5(d)) is apparently more complex and it does not reveal such a clear computational pattern. It has also learned the 'carry' to the first and second output bits (the hidden neurons labeled by 5 and 9, respectively), but the functions of other hidden and output neurons are less obvious.

5.5.6 Experiment 5.5.6 – Sensitivity analysis

Experiment setting

In Experiment 5.5.6, we evaluated, in what way do the analytical and approximative methods for sensitivity control affect the overall network sensitivity. They are designed to make the sensitivity coefficients smaller in general. Another question is, whether they also increase the differences among the achieved sensitivity coefficients of the respective input features. We also assessed, which input features were evaluated as the most important ones based on the sensitivity analysis of BP-networks trained by the SCG, SCGIR, SCGS and SCGSA methods.

The experiment setting is based on the setting of Experiment 5.2.1 described on page 123. The test involved the *World Bank* task (data set **WB**). All methods were applied with pruning of both input and hidden neurons to the same set of 100 randomly initialized BP-networks with the topologies 35-50-5, the parameter *maxEpochs* was set to 1101. The setting of parameters c_F and c_G for the respective training algorithms is stated in Table 5.24.

Table 5.24: Experiment 5.5.6 – Setting of the parameters c_F and c_G.

parameter	method			
	SCG	SCGIR	SCGS	SCGSA
c_F	–	10^{-6}	10^{-6}	10^{-6}
c_G	–	–	$5 \cdot 10^{-7}$	10^{-5}

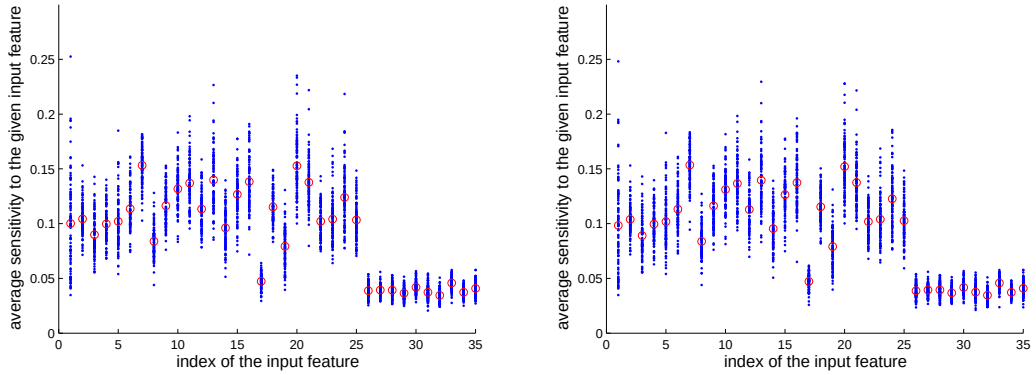
For each trained BP-network and each input feature u, we computed the values of absolute sensitivity coefficients averaged over all network outputs v and all training patterns p:

$$S_u = \underset{\{v,p\}}{\text{mean}} |S_{uv}^p|, \tag{5.4}$$

where the sensitivity coefficients S_{uv}^p are defined by Equations (4.16) and (4.17). The values of S_u for the respective training algorithms are stated in Figures 5.6 (for BP-networks before pruning) and 5.7 (for BP-networks after pruning and retraining). Figure 5.8 gives a summary of the values of S_u averaged over 100 random network initializations. Table 5.25 contains the most and least important input features based on S_u.

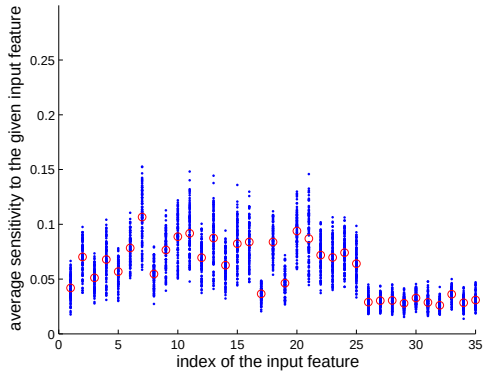
Results – On the overall sensitivity of the networks

Figures 5.6, 5.7, 5.8 confirm that the SCGS and SCGSA methods remarkably decrease the absolute values of sensitivity coefficients, when compared to SCG. The SCGS method apparently outperforms SCGSA in this respect. The sensitivity coefficients achieved for the SCGIR-trained networks are about the same as for the SCG-trained networks. On the other hand, the differences among the

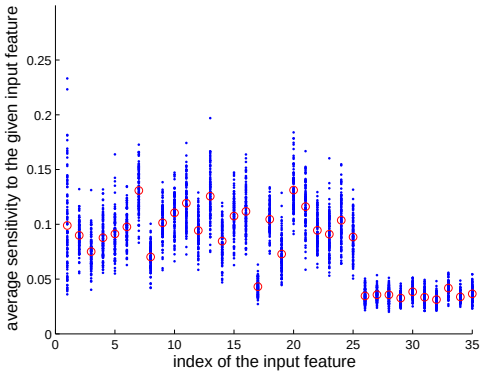


(a) SCG

(b) SCGIR



(c) SCGS



(d) SCGSA

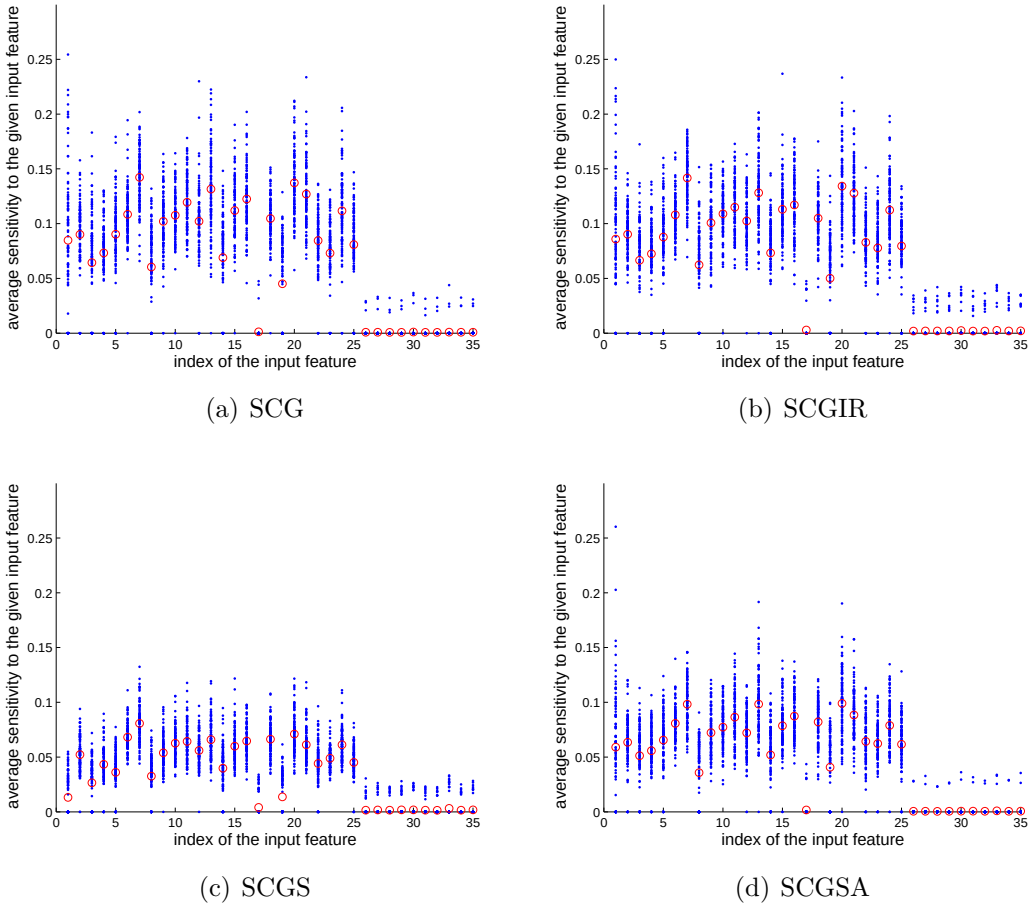
Figure 5.6: Experiment 5.5.6 – Average values of the absolute sensitivity coefficients for BP-networks with the topologies 35-50-5 trained by the SCG, SCGIR, SCGS and SCGSA methods on the **WB** data set (before pruning). Blue points correspond to the values of $S_u = \text{mean}_{\{v,p\}} |S_{uv}^p|$ for all input features indexed by u, where p and v index all training patterns and network outputs, respectively. Average values of S_u over 100 network initializations are indicated by red circles.

sensitivity coefficients corresponding to the respective input features seem to be proportional for all of the methods.

Results – On the sensitivity of the networks to the input features

Table 5.25 shows that all of the methods clearly identified and pruned the redundant input features 26,...,35. They also agreed on the two most important input features 7 (GINI-Index) and 20 (Internet users). Figure 1 in the Introduction of this thesis shows the mutual dependence of the input features 15 (Fertility rate), 16 (Fixed line and mobile phone subscribers), 20 (Internet users) and 21 (Life expectancy at birth). These input features, that give a very similar information, are among the 9 most important ones based on S_u for all of the tested methods.

The outputs of the SCGS- and SCGSA-trained BP-networks are sensitive most to slightly different input features than the outputs of the SCG- and SCGIR-trained ones. For example, the SCGS method emphasize the input features 11, 10 over 13, 16 and 18 over 24 (see Table 5.1 for a detailed description of these



(a) SCG (b) SCGIR

(c) SCGS (d) SCGSA

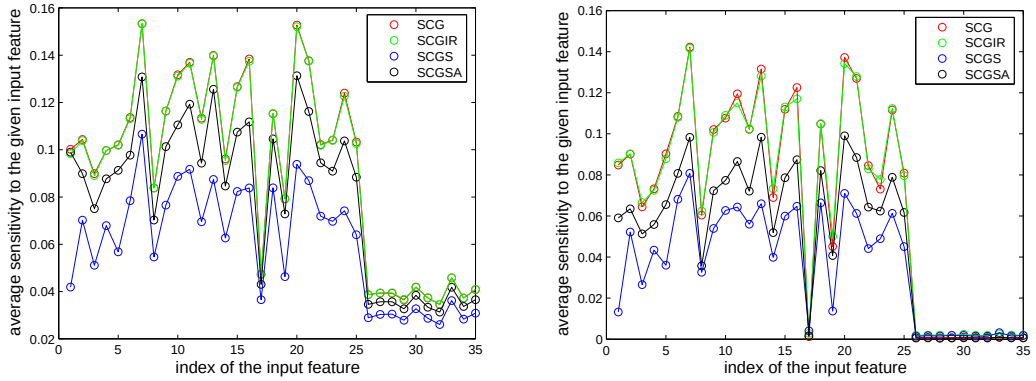
Figure 5.7: Experiment 5.5.6 – Average values of the absolute sensitivity coefficients for BP-networks with the topologies 35-50-5 trained by the SCG, SCGIR, SCGS and SCGSA methods on the **WB** data set (after pruning and retraining). Blue points correspond to the values of $S_u = \text{mean}_{\{v,p\}} |S^p_{uv}|$ for all input features indexed by u, where p and v index all training patterns and network outputs, respectively. Average values of S_u over the 100 network initializations are indicated by red circles.

input features). This fact together with improved generalization of the SCGS- and SCGSA-trained BP-networks support the idea, that both SCGS and SCGSA favor different, smoother network functions than SCG and SCGIR.

5.5.7 Summary of Structure optimization

In sum, our training-and-pruning methodology together with the SCGS and SCGSA training algorithms proved to reliably identify redundant input features and to develop BP-networks with a simple structure and improved generalization. It outperforms in these respects both pruning based on alternative relevance measures and the cluster- and sensitivity-based feature selection techniques.

The SCGS and SCGSA methods applied together with pruning remarkably outperform the SCG and SCGWD methods in their generalization abilities. They are also more robust to the initial network topology and to the presence of redundant input features.



(a) before pruning (b) after pruning

Figure 5.8: Experiment 5.5.6 – Average values of the absolute sensitivity coefficients for BP-networks with the topologies 35-50-5 trained by the SCG, SCGIR, SCGS and SCGSA methods on the **WB** data set (before and after pruning). The stated values of S_u correspond to the mean over 100 random network initializations.

Table 5.25: Experiment 5.5.6 – The most and least important input features for BP-networks with the topologies 35-50-5 trained by the respective methods on the **WB** data set.

The most important input features:				
order	SCG	SCGIR	SCGS	SCGSA
1.	7	7	7	20
2.	20	20	20	7
3.	13	13	11	13
4.	16	21	10	11
5.	21	16	13	21
6.	11	11	21	16
7.	10	10	18	10
8.	15	15	16	15
9.	24	24	15	18

The least important input features:				
order	SCG	SCGIR	SCGS	SCGSA
21.	14	14	8	14
22.	3	3	3	3
23.	8	8	19	19
24.	19	19	1	8
25.	17	17	17	17
26. – 35.	26 – 35	26 – 35	26 – 35	26 – 35

Conclusions

The main subject of this thesis has been the computational model of BP-networks. Our goal was to use this model to find complex non-linear dependencies in the data as quickly as possible and to describe them as simply as possible. For this reason, we investigated the ability of BP-networks to create during training a robust, transparent and simple internal structure. At the same time, we focused on fast training, adequate generalization and robustness of the model to noise and tunable parameters.

To achieve our goal, we developed a general framework for training of BP-networks. Our framework takes advantage of several existing and newly proposed techniques for BP-network training. It is based on the general schema of the fast and robust SCG-training algorithm [75] (Algorithm 4.2 on page 80). The SCG algorithm is enhanced by three regularization techniques – IR (for learning internal representation during training, described in Section 4.1.2), SC (for analytical sensitivity control, described in Section 4.2.4), and SCA (for approximative sensitivity control, described in Section 4.3.2).

The IR-regularization technique [86] forces BP-networks to create a condensed internal representation during training. In this way, it contributes to transparent and simple internal structure of the trained BP-network and simplifies pruning based on internal representation. It also facilitates knowledge extraction from the model.

The new-proposed regularization techniques, SC and SCA, focus on sensitivity inhibition during training. While the analytical SC-regularization technique is very complex and makes the training process extremely time consuming, its approximative alternative, SCA, is very fast. Despite its approximative nature, SCA keeps most of the advantages of the SC-method. Both methods contribute to smoother network function and better generalization ability of the trained BP-networks. They also simplify pruning based on sensitivity analysis and support an easy interpretation of the extracted knowledge.

As an integral part of our framework, we developed a general training-and-pruning methodology based on internal representation and sensitivity analysis (Algorithms 4.3 and 4.4). It enables to identify relevant input features and find a suitable topology of the BP-network automatically during training. In such a way it simplifies all of the following processes: preprocessing, parameter-tuning, training and knowledge extraction from the trained BP-network. It also remarkably improves the generalization capability of the trained model.

In addition to the above-listed key methods, our framework contains also further supplementary methods, that support its good qualities (e.g., learning from hints [106] and the early stopping strategy [95]).

The introduced framework was experimentally evaluated on several data sets of various properties (i.e., discrete and continuous, artificial and real-world). In the experiments performed, we assessed advantages and weaknesses of the entire framework and of the included methods when compared with alternative techniques. We considered related methods for BP-network training, regularization, feature selection and pruning.

An advantage of the techniques included in our framework is, that they can be combined in more ways. During our research, we successively introduced three

versions of our framework – SCGIR (Algorithm 4.1 on page 79), SCGS (Algorithm 4.5 on page 98) and SCGSA (Algorithm 4.6 on page 111). These training algorithms differ primarily in the regularization techniques included. The earliest method, SCGIR, comprises the IR-regularization technique and learning from hints. The SCGS and SCGSA methods combine the IR-regularization technique with one of the sensitivity-inhibiting techniques (SC for SCGS and SCA for SCGSA). The benefits and drawbacks of the respective training algorithms, based on the experiments performed so far, are summarized in Table 5.26.

Table 5.26: Summary of the qualities of the proposed methods (SCGIR, SCGS and SCGSA), when compared to the standard SCG training algorithm. The performance of the methods based on the respective criteria is in each column indicated by the number of stars (1-5) — the best-performing method is labeled by '*****'.

Method	Generalization	Speed	Robustness to parameters	Transparency	Structure optimization
SCG	*** satisfactory	***** fast	***** robust	* weak	* none
SCGIR	*** comparable with SCG	**** maximally two times slower than SCG	*** relatively sensitive to parameter c_F	***** strong enforcement for greater values of c_F	*** pruning of hidden neurons
SCGS	***** bigger improvements for discrete data and for networks with more hidden layers	* extremely slow	***** robust to parameters c_F and c_G	**** medium, but stable enforcement	***** pruning of hidden and input neurons
SCGSA	***** bigger improvements for discrete data and for networks with more hidden layers	**** maximally two times slower than SCG	***** robust to parameters c_F and c_G	**** medium, but stable enforcement	***** pruning of hidden and input neurons

The firstly-introduced method, SCGIR, proved in our experiments to be a very fast training algorithm with an outstanding ability to form a condensed internal representation during training. It is maximally two times slower than

SCG, while it yields networks with a comparable generalization ability. Its capability to create a transparent network structure is similar to GDIR. Anyway, the right choice of the trade-off coefficient c_F applied during training can impact the quality of the solution obtained. Too large values of c_F might result into BP-networks with a perfectly formed condensed internal representation yet incapable of approximating the desired function because of saturated hidden neuron outputs.

The SCGS and SCGSA methods confirmed in our experiments to perform comparably with each other in many aspects. When considering the generalization ability of the trained BP-networks and their sensitivity to noise in the data, both SCGS and SCGSA remarkably outperformed SCG, SCGIR and other techniques traditionally used to improve network's generalization (e.g., training with jitter [95]). On the other hand, the actual behavior of the SCGS and SCGSA methods seems to depend on the character of the processed data and on the architecture of the trained network. Based on the experiments performed so far, higher improvement might be achieved for discrete data and for networks with more hidden layers.

Although SCGS and SCGSA are slightly less powerful in forming condensed internal representation than SCGIR, they are more robust to the right choice of the trade-off coefficients (c_F, c_G). Thanks to the introduced training-and-pruning methodology, both SCGS and SCGSA methods also reliably identified redundant input features and developed BP-networks with a relatively simple structure. They outperformed in these respects alternative pruning and feature selection techniques.

Based on the experiments performed so far, the last-introduced method, SCGSA, seems to represent the best choice for practical applications. According to most of the tested criteria, the results for SCGSA are comparable or better than for SCGIR. When comparing SCGSA to SCGS, its main advantage consists in its low time costs. Contrary to SCGS, which is very time expensive, SCGSA is maximally two times slower than SCG.

In sum, the main outcome of this thesis consist in developing a general framework for training of BP-networks with the following advantages:

1. Improved generalization ability of the trained BP-networks and their lower sensitivity to noise in the data.
2. Fast training, robust to tunable parameters.
3. Creation of a simpler and more transparent internal network structure (due to enforcement of condensed internal representation and sensitivity inhibition during training).
4. Robustness of the model to initial topology and redundant input features (due to sophisticated pruning of hidden and input neurons).
5. Simplified knowledge extraction from the model and easier interpretation of the extracted knowledge.

Future work

Above all, we would like to test our framework more extensively on larger data sets comprising several thousands of patterns and several thousands of input

features, in order to provide a greater number of statistically significant results.

In the past years, a strong effort has been made in the field of so-called deep networks (introduced by LeCun et al. in [65]). Deep networks succeed in some demanding tasks such as image processing, speech recognition or machine translation, where the standard low-layered BP-networks fail. An interesting and challenging task would be to find a way, how to successfully apply the main principles proposed in this thesis to the model of deep networks. Especially sensitivity inhibition and sensitivity-based pruning might simplify training and contribute to improved robustness, stability and generalization of this model.

Bibliography

[1] Abu-Mostafa, Yaser S. A Method for Learning From Hints. In: *Advances in Neural Information Processing Systems*, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 1993. Vol. 5, pp. 73–80. ISBN 1-558-60274-7.

[2] Abu-Mostafa, Yaser S. Hints and the VC dimension. *Neural Computing*. Vol. 5, No. 2, pp. 278–288, 1993.

[3] Abu-Mostafa, Yaser S. Learning from Hints. *Journal of Complexity*. Vol. 10, No. 1, pp. 165 – 178, 1994.

[4] Alippi, C., Piuri, V. and Sami, M. Sensitivity to errors in artificial neural networks: a behavioral approach. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*. Vol. 42, No. 6, pp. 358–361, June 1995.

[5] Andrejková, Gabriela. Applications of the approximation theory by neural Networks. *Neural Network World*. Vol. 5, pp. 787–795, 2000.

[6] Andrejková, Gabriela. Incremental Approximation by Layer Neural Networks. In: *The State of the Art in Computational Intelligence : Proceedings of the European Symposium on Computational Intelligence*, Berlin / Heidelberg, Germany. Springer-Verlag, 2000. ISCI'2010, pp. 15–20.

[7] Attik, Mohammed, Bougrain, Laurent and Alexandre, Frederic. Neural Network Topology Optimization. In: *International Conference on Artificial Neural Networks*, Berlin / Heidelberg, Germany. Springer, 2005. Vol. 3697 of *ICANN'05*, pp. 53–58.

[8] Bache, K. and Lichman, M. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, 2013. Available from: `<http://archive.ics.uci.edu/ml>`.

[9] Baird, Sarah, Friedman, Jed and Schady, Norbert. Aggregate Income Shocks and Infant Mortality in the Developing World. *The Review of Economics and Statistics*. Vol. 93, No. 3, pp. 847–856, 2011.

[10] Baldi, Pierre and Hornik, Kurt. Learning in Linear Neural Networks: a Survey. *IEEE Transactions on Neural Networks*. Vol. 6, pp. 837–858, 1995.

[11] Battini, R. and Masulli, F. BFGS Optimization for Faster and Automated Supervised Learning. In: *International Neural Network Conference*, Dordrecht, Germany. Kluwer, 1990. Vol. 2 of *INCC 90*, pp. 757–760.

[12] Battiti, Roberto. First and Second-Order Methods for Learning: between Steepest Descent and Newton's Method. *Neural Computation*. Vol. 4, pp. 141–166, 1992.

[13] Baum, Eric B. and Haussler, David. What Size Net Gives Valid Generalization? *Neural Computation*. Vol. 1, No. 1, pp. 151–160, March 1989. ISSN 0899-7667.

[14] Beale, E.M.L. A Derivation of Conjugate Gradients. In: Lootsma, F.A., ed. *Numerical Methods in Nonlinear Optimization*. London, UK: Academic Press, 1997. chapter 4, pp. 39–43.

[15] Bishop, Christopher M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006. ISBN 0-387-31073-8.

[16] Bishop, Christopher M. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995. ISBN 0-198-53864-2.

[17] Blum, Avrim L. and Langley, Pat. Selection of relevant features and examples in machine learning. *Artificial Intelligence*. Vol. 97, pp. 245–271, 1997.

[18] Butterworth, Richard, Piatetsky-Shapiro, Gregory and Simovici, Dan A. On Feature Selection Through Clustering. In: *Proceedings of the Fifth IEEE International Conference on Data Mining*, Washington, DC, USA. IEEE Computer Society, 2005. ICDM'05, pp. 581–584. ISBN 0-769-52278-5.

[19] Castillo, Enrique F., Guijarro-Berdiñas, Bertha, Fontenla-Romero, Oscar and Alonso-Betanzos, Amparo. A Very Fast Learning Method for Neural Networks Based on Sensitivity Analysis. *Journal of Machine Learning Research*. Vol. 7, pp. 1159–1182, 2006.

[20] Chan, Lai-Wan. Levenberg-Marquardt Learning and Regularization. In: *Progress in Neural Information Processing*. Springer-Verlag, 1996. pp. 139–144.

[21] Charalambous, C. Conjugate gradient algorithm for efficient training of artificial neural networks. *Proceedings of the IEEE*. Vol. 139, No. 3, pp. 301–310, 1992.

[22] Choi, Jin-Young and Choi, Chong-Ho. Sensitivity analysis of multilayer perceptron with differentiable activation functions. *IEEE Transactions on Neural Networks*. Vol. 3, No. 1, pp. 101–107, 1992.

[23] Christiansen, Morten H. Improving learning and generalization in neural networks through the acquisition of multiple related functions. In: Bullinaria, J.A., Glasspool, D.G. and Houghton, G., eds. *Fourth Neural Computation and Psychology Workshop: Connectionist Representations*, London, UK. Springer-Verlag, 1998. pp. 58–70.

[24] Covões, Thiago F., Hruschka, Eduardo R., Castro, Leandro N. and Santos, Átila M. A Cluster-Based Feature Selection Approach. In: Corchado, Emilio, Wu, Xindong, Oja, Erkki, Herrero, Álvaro and Baruque, Bruno, eds. *Hybrid Artificial Intelligence Systems*, Vol. 5572 of *Lecture Notes in Computer Science*. Berlin / Heidelberg, Germany: Springer, 2009. pp. 169–176.

[25] Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*. Vol. 2, pp. 303–314, 1989.

[26] Demuth, Howard, Beale, Mark and Hagan, Martin. *Neural Network Toolbox 5: Users Guide*. The Mathworks, Inc., 2007.

[27] Engelbrecht, A. and Cloete, I. A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks. In: *IEEE International Conference in Neural Networks*, Washington, DC, USA. 1996. Vol. 2 of *IEEE ICNN'96*, pp. 1274–1277.

[28] Engelbrecht, A. P. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*. Vol. 12, No. 6, pp. 1386–1399, 2001.

[29] Engelbrecht, A. P., Cloete, Ian and Zurada, Jacek M. Determining the Significance of Input Parameters using Sensitivity Analysis. In: *International Work-Conference on Artificial Neural Networks*, 1995. IWANN'95, pp. 382–388.

[30] Engelbrecht, Andries Petrus. Sensitivity Analysis for Selective Learning by Feedforward Neural Networks. *Fundamenta Informaticae*. Vol. 45, No. 4, pp. 295–328, 2001.

[31] Engelbrecht, Andries Petrus. Sensitivity Analysis for Decision Boundaries. *Neural Processing Letters*. Vol. 10, No. 3, pp. 253–266, 1999.

[32] Fahlman, Scott E. Faster-Learning Variations on Back-Propagation: An Empirical Study. In: *Proceedings of the Connectionist Models Summer School*, Los Altos, CA, USA. Morgan-Kaufmann, 1988.

[33] Fahlman, Scott E. and Lebiere, Christian. The Cascade-Correlation Learning Architecture. In: Touretzky, David S., ed. *Advances in Neural Information Processing Systems*, Vol. 2. San Francisco, CA, USA: Morgan Kaufmann, 1990. pp. 524–532.

[34] Fidalgo, J. N. Feature subset selection based on ANN sensitivity analysis - a practical study. *Advances in Neural Networks and Applications*. pp. 206–211, 2001.

[35] Fletcher, R. *Practical methods of optimization*. A Wiley Interscience Publication, 2nd edition, 1987.

[36] Garson, G. David. Interpreting neural-network connection weights. *AI Expert*. Vol. 6, No. 4, pp. 46–51, 1991. ISSN 0888-3785.

[37] Ghosh, Ranadhir, Ghosh, Moumita, Yearwood, John and Bagirov, Adil. Comparative Analysis of Genetic Algorithm, Simulated Annealing and Cutting Angle Method for Artificial Neural Networks. In: *Proceedings of the 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*, Berlin / Heidelberg, Germany. Springer-Verlag, 2005. MLDM'05, pp. 62–70. ISBN 3-540-26923-1.

[38] Gällmo, O. and Carlström, J. Some Experiments Using Extra Output Learning to Hint Multi Layer Perceptrons. In: Niklasson, L.F. and Boden, M.B., eds. *Current Trends in Connectionism*, Hillsdale, MI, USA. 1995. SCC'95, pp. 179–190.

[39] Guijarro-Berdiñas, Bertha, Fontenla-Romero, Oscar, Pérez-Sánchez, Beatriz and Alonso-Betanzos, Amparo. A Regularized Learning Method for Neural Networks Based on Sensitivity Analysis. In: *ESANN*, 2008. pp. 289–294.

[40] Guyon, Isabelle. An introduction to variable and feature selection. *Journal of Machine Learning Research*. Vol. 3, pp. 1157–1182, 2003.

[41] Guyon, Isabelle, Gunn, Steve, Nikravesh, Masoud and Zadeh, Lotfi A. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., 2006. ISBN 3-540-35487-5.

[42] Hagan, Martin T., Demuth, Howard B. and Beale, Mark. *Neural Network Design*. PWS Publishing Co., 1996. ISBN 0-534-94332-2.

[43] Hagiwara, Masafumi. A simple and effective method for removal of hidden units and weights. *Neurocomputing*. Vol. 6, No. 2, pp. 207–218, 1994.

[44] Hancock, P. J. B. Pruning Neural Nets by Genetic Algorithm. In: Aleksander, I. and Taylor, J.G., eds. *International Conference on Artificial Neural Networks*, Brighton, UK. Elsevier, 1992. pp. 991–994.

[45] Hanson, Stephen José and Pratt, Lorien. Comparing biases for minimal network construction with back-propagation. *Advances in Neural Information Processing Systems*. Vol. 1, pp. 177–185, 1989.

[46] Hara, Kazuyuki and Okada, Masato. Online learning of a simple perceptron learning with margin. *Systems and Computers in Japan*. Vol. 35, No. 7, pp. 98–105, 2004.

[47] Hassibi, B., Stork, D. G. and Wolf, G. J. Optimal Brain Surgeon and general network pruning. In: *IEEE International Conference on Neural Networks*, San Francisco, CA, USA. 1993. Vol. 1 of *IEEE ICNN'93*, pp. 293–299.

[48] Haykin, Simon. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 2nd edition, 1998. ISBN 0-132-73350-1.

[49] Holmström, Lasse and Koistinen, Petri. Using Additive Noise in Back-Propagation Training. Research Reports A3, Rolf Nevanlinna Institute, 1990.

[50] Hornik, Kurt. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*. Vol. 4, No. 2, pp. 251–257, March 1991. ISSN 0893-6080.

[51] Hunt, S. D. and Deller, John R. Selective training of feedforward artificial neural networks using matrix perturbation theory. *Neural Networks*. Vol. 8, No. 6, pp. 931–944, 1995.

[52] Jacobs, Robert A. Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks*. Vol. 1, pp. 295–307, 1988.

[53] Jolliffe, I.T. *Principal Component Analysis*. Springer-Verlag, 2nd edition, 2002.

[54] Judd, J. Stephen. *Neural network design and the complexity of learning*. Neural network modeling and connectionism. MIT Press, 1990.

[55] Kantardzic, Mehmed. *Data Mining: Concepts, Models, Methods and Algorithms*. John Wiley & Sons, Inc., 2002. ISBN 0-471-22852-4.

[56] Karnin, E. D. A simple procedure for pruning back-propagation trained neural networks. In: *IEEE International Conference on Neural Networks*, 1990. Vol. 1, pp. 239–242.

[57] Koda, M. Neural network learning based on stochastic sensitivity analysis. *IEEE Transactions on Systems, Man and Cybernetics, Part B*. Vol. 27, No. 1, pp. 132–135, 1997.

[58] Kohavi, Ron and John, George H. Wrappers for Feature Subset Selection. *Artificial Intelligence*. Vol. 97, No. 1, pp. 273–324, 1997.

[59] Kůrková, Věra. Approximation of functions by perceptron networks with bounded number of hidden units. *Neural Networks*. Vol. 8, No. 5, pp. 745 – 750, 1995. ISSN 0893-6080.

[60] Lamers, M.H., Kok, J.N. and Lebret, E. A multilevel nonlinearity study design. In: *IEEE World Congress on Computational Intelligence Neural Networks*, May 1998. Vol. 1, pp. 730–734.

[61] Lampinen, Jouko, Litkey, Paula and Hakkarainen, Harri. Selection of Training Samples for Learning With Hints. In: *International Joint Conference on Neural Networks*, Washington, DC, USA. 1999. IJCNN'99.

[62] LeCun, Y., Denker, J., Solla, S., Howard, R. E. and Jackel, L. D. Optimal Brain Damage. In: Touretzky, D. S., ed. *Advances in Neural Information Processing Systems*, San Mateo, CA, USA. Morgan Kauffman, 1990. Vol. 2.

[63] LeCun, Yann. Une procédure d'apprentissage pour réseau à seuil asymétrique. In: *Proceedings of Cognitiva 85*, Paris, France. 1985. pp. 599–604.

[64] Lecun, Yann, Simard, Patrice Y. and Pearlmutter, Barak. Automatic Learning Rate Maximization by On-Line Estimation of the Hessian's Eigenvectors. In: *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1993. Vol. 5, pp. 156–163.

[65] LeCun, Yann, Bottou, Léon, Bengio, Yoshua and Haffner, Patrick. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*. Vol. 86, No. 11, pp. 2278–2324, 1998.

[66] Lee, Tsu-Chang. *Structure Level Adaptation for Artificial Neural Networks*. Kluwer Academic Publishers, 1991. ISBN 0-792-39151-9.

[67] Levenberg, K. A Method for the Solution of Certain Problems in Least Squares. In: *Quarterly Applied Math*, 1944. Vol. 2, pp. 164–168.

[68] Liu, Quanjin, Zhao, Zhimin and Wang, Yong. Study on Feature Selection Based on Fuzzy Clustering Algorithm. In: Qian, Zhihong, Cao, Lei, Su, Weilian, Wang, Tingkai and Yang, Huamin, eds. *Recent Advances in Computer Science and Information Engineering*, Vol. 124 of *Lecture Notes in Electrical Engineering*. Berlin / Heidelberg, Germany: Springer, 2012. pp. 155–161.

[69] Liu, Yinyin, Starzyk, Janusz A. and Zhu, Zhen. Optimized Approximation Algorithm in Neural Networks Without Overfitting. *IEEE Transactions on Neural Networks*. Vol. 19, No. 6, pp. 983–995, 2008.

[70] MacQueeen, J. B. Some Methods for Classification and Analysis of Multivariate Observations. In: *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkley, CA, USA. 1967. Vol. 1, pp. 281–297.

[71] Marquardt, Donald W. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*. Vol. 11, No. 2, pp. 431–441, 1963.

[72] MATLAB and Neural Network Toolbox. *Version 7.0.1 (R14)*. The MathWorks, Inc., 2004.

[73] Minnix, J.I. Fault tolerance of the backpropagation neural network trained on noisy inputs. In: *International Joint Conference on Neural Networks*, June 1992. Vol. 1 of *IJCNN'92*, pp. 847–852.

[74] Mitchell, Thomas M. *Machine Learning*. McGraw-Hill, Inc., 1st edition, 1997. ISBN 0-070-42807-7.

[75] Møller, Martin F. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*. Vol. 6, No. 4, pp. 525–533, 1993.

[76] Montaño, J. J. and Palmer, A. Numeric sensitivity analysis applied to feedforward neural networks. *Neural Computing and Applications*. Vol. 12, No. 2, pp. 119–125, 2003.

[77] Mozer, Michael C. and Smolensky, Paul. Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Advances in Neural Information Processing Systems*. Vol. 1, pp. 107–115, 1989.

[78] Mrázová, Iveta and Reitermanová, Zuzana. Enforced knowledge extraction with BP-networks. In: Dagli, Cihan, ed. *Intelligent Engineering Systems through Artificial Neural Networks*, New York, NY, USA. ASME Press, 2007. Vol. 17, pp. 285–290. ISBN 0-791-80265-5.

[79] Mrázová, Iveta. Controlled Learning of GREN-networks. In: Dagli, C. H., Buczak, A. L., Ghosh, J., Embrechts, M. J., Ersoy, O. and Kercel, S., eds. *Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining and Complex Systems*. ASME Press, 2001. ASME Press Series, pp. 21–26.

[80] Mrázová, Iveta. *Knowledge Extraction with Neural Networks: Significant Patterns and their Representation in Back-Propagation Networks*. LAP LAMBERT Academic Publishing, 2011.

[81] Mrázová, Iveta and Petříčková, Zuzana. Fast Sensitivity-Based Training of BP-Networks. In: *Artificial Neural Networks and Machine Learning*. Springer, 2014. Vol. 8681 of *Lecture Notes in Computer Science*, pp. 507–514.

[82] Mrázová, Iveta and Reitermanová, Zuzana. Enforced knowledge extraction with BP-networks. Technical Report 2007/7, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague, July 2007.

[83] Mrázová, Iveta and Reitermanová, Zuzana. A new sensitivity-based pruning technique for feed-forward neural networks that improves generalization. In: *International Joint Conference on Neural Networks*. IEEE, 2011. IJCNN'2011, pp. 2143–2150.

[84] Mrázová, Iveta and Reitermanová, Zuzana. Sensitivity-based SCG-training of BP-networks. *Procedia Computer Science – Complex adaptive systems*. Vol. 6, No. 0, pp. 177 – 182, 2011. ISSN 1877-0509.

[85] Mrázová, Iveta and Reitermanová, Zuzana. A new sensitivity-based pruning technique for feed-forward neural networks that improves generalization. Technical Report 2011/3, Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University in Prague, March 2011.

[86] Mrázová, Iveta and Wang, Dianhui. Improved generalization of neural classifiers with enforced internal representation. *Neurocomputing*. Vol. 70, No. 16-18, pp. 2940–2952, 2007.

[87] Nguyen, Derrick and Widrow, Bernard. Improving the Learning Speed of 2-layer Neural Networks by Choosing. In: *Initial Values of the Adaptive Weights, International Joint Conference of Neural Networks*, 1990. pp. 21–26.

[88] Oh, Sang-Hoon and Lee, Youngjik. Sensitivity analysis of single hidden-layer neural networks with threshold functions. *IEEE Transactions on Neural Networks*. Vol. 6, No. 4, pp. 1005–1007, 1995.

[89] Parker, D. B. Learning-Logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT, 1985.

[90] Polak, E. and Ribiere, G. Note sur la convergence de methodes de directions conjugees. *Francaise Informate Recherche Operatonelle*. Vol. 3, pp. 35–43, 1969.

[91] Powell, M. J. D. Restart procedures for the conjugate gradient method. *Mathematical Programming*. Vol. 2, No. 1, pp. 241–254, 1977. ISSN 0025-5610.

[92] Procházka, Aleš and Pavelka, Aleš. Feed-forward and recurrent neural networks in signal prediction. In: *5th IEEE International Conference on Computational Cybernetics*. IEEE, 2007. pp. 93–96.

[93] Reed, R., Marks, II R.J. and Oh, S. Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter. *IEEE Transactions on Neural Networks*. Vol. 6, No. 3, pp. 529–538, May 1995. ISSN 1045-9227.

[94] Reed, Russell. Pruning Algorithms – A survey. *IEEE Transactions on Neural Networks*. Vol. 4, pp. 740–747, 1993.

[95] Reed, Russell D. and Marks, Robert J. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, 1998. ISBN 0-262-18190-8.

[96] Reed, Russell D. and Marks, Robert J. Neurosmithing: Improving Neural Network Learning. In: Arbib, Michael A., ed. *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998. pp. 639–644. ISBN 0-262-51102-9.

[97] Reitermanová, Zuzana. Knowledge Extraction with BP-netwoks. Master's thesis, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague, 2007.

[98] Reunanen, Juha, Guyon, Isabelle and Elisseeff, Andre. Overfitting in Making Comparisons Between Variable Selection Methods. *Journal of Machine Learning Research*. Vol. 3, pp. 1371–1382, 2003.

[99] Riedmiller, Martin. Advanced supervised learning in multi-layer perceptrons — From backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*. Vol. 16, No. 3, pp. 265 – 278, 1994.

[100] Rojas, Raul. *Neural Networks - A Systematic Introduction*. Springer-Verlag, 1996.

[101] Rosenblatt, F. The Perceptron: A probabilistic model for information storage and organisation in the brain. *Psychological Review*. Vol. 65, No. 2, pp. 368–408, 1958.

[102] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1, pp. 318–362, 1986.

[103] Sharkey, Noel E. Connectionist representation techniques. *Artificial Intelligence Review*. Vol. 5, No. 3, pp. 143–167, 1991.

[104] Sietsma, J. and Dow, R. J. F. Creating artificial neural networks that generalize. *Neural Networks*. Vol. 4, pp. 67–79, 1991.

[105] Silva, Fernando M. and Almeida, Luis B. Acceleration Techniques for the Backpropagation Algorithm. In: Almeida, Luis B. and Wellekens, Christian, eds. *EURASIP Workshop*. Springer, 1990. Vol. 412 of *Lecture Notes in Computer Science*, pp. 110–119.

[106] Suddarth, S. C. and Kergosien, Y. L. Rule-Injection Hints as a Means of Improving Network Performance and Learning Time. In: *EURASIP Workshop 1990 on Neural Networks*, London, UK. Springer-Verlag, 1990. Vol. 412, pp. 120–129.

[107] Šíma, Jiří. Training a Single Sigmoidal Neuron Is Hard. *Neural Computation*. Vol. 14, No. 11, pp. 2709–2728, 2002.

[108] Tchaban, T., Taylor, M. J. and Griffin, J. P. Establishing impacts of the inputs in a feedforward neural network. *Neural Computing and Applications*. Vol. 7, No. 4, pp. 309–317, 1998.

[109] The World Bank Group. *World Development Report 2007/2008*. Oxford University Press, 2008.

[110] Thimm, Georg and Fiesler, Emile. Neural Network Initialization. In: Mira, José and Hernández, Francisco Sandoval, eds. *International Work-Conference on Artificial Neural Networks*. Springer, 1995. Vol. 930 of *Lecture Notes in Computer Science*, pp. 535–542. ISBN 3-540-59497-3.

[111] Tsaih, R. Sensitivity analysis, neural networks, and the finance. In: *International Joint Conference on Neural Networks*, 1999. Vol. 6 of *IJCNN'99*, pp. 3830–3835.

[112] Tuv, Eugene, Borisov, Alexander, Runger, George and Torkkola, Kari. Feature Selection with Ensembles, Artificial Variables, and Redundancy Elimination. *Journal of Machine Learning Research*. Vol. 10, pp. 1341–1366, December 2009. ISSN 1532-4435.

[113] Vapnik, V. N. and Chervonenkis, A. Ya. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and its Applications*. Vol. 16, No. 2, pp. 264–280, 1971.

[114] Vapnik, Vladimir Naoumovitch. *The nature of statistical learning theory*. Statistics for engineering and information science. Springer, 2000. ISBN 0-387-98780-0.

[115] Wang, W., Jones, P. and Partridge, D. Assessing the Impact of Input Features in a Feedforward Neural Network. *Neural Computing and Applications*. Vol. 9, No. 2, pp. 101–112, 2000.

[116] Wasserman, P. D. Experiments in Translating Chinese Characters Using Backpropagation. In: *COMPCON*. IEEE Computer Society, 1988. pp. 399–402.

[117] Weigend, Andreas S., Rumelhart, David E. and Huberman, Bernardo A. Generalization by Weight-Elimination with Application to Forecasting. In: Lippmann, Richard P., Moody, John E. and Touretzky, David S., eds. *Advances in Neural Information Processing Systems*, Vol. 3. San Francisco, CA, USA: Morgan Kaufmann, 1991. pp. 875–882.

[118] Werbos, P. J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[119] Werbos, P. J. Back-propagation: Past and future. In: *IEEE International Conference on Neural Networks*, New York, NY, USA. IEEE Press, 1988. Vol. 1, pp. 343–353.

[120] Whitley, D., Starkweather, T. and Bogart, C. Genetic algorithms and neural networks: Optimizing connections and connectivity. In: *Parallel Computing*, 1990. Vol. 14, pp. 347–361.

[121] Xu, Lei. Data smoothing regularization, multi-sets-learning, and problem solving strategies. *Neural Networks*. Vol. 16, No. 5-6, pp. 817–825, 2003.

[122] Yeh, I-Cheng and Cheng, Wei-Lun. First and Second Order Sensitivity Analysis of MLP. *Neurocomputing*. Vol. 73, No. 10-12, pp. 2225–2233, June 2010. ISSN 0925-2312.

[123] Yeung, Daniel S, Cloete, Ian, Shi, Daming and Ng, Wing WY. *Sensitivity Analysis for Neural Networks*. Natural Computing Series. Springer, 2010.

[124] Yu, Yeong-H. and Simmons, Robert F. Extra Output Biased Learning. In: *International Joint Conference on Neural Networks*, San Diego, CA, USA. University of Texas at Austin, 1990. Vol. 3 of *IJCNN'90*, pp. 161–166.

[125] Zhong, Shuiming, Zeng, Xiaoqin, Wu, Shengli and Han, Lixin. Sensitivity-Based Adaptive Learning Rules for Binary Feedforward Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*. Vol. 23, No. 3, pp. 480–491, 2012.

[126] Zurada, Jacek M., Malinowski, Aleksander and Cloete, Ian. Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network. In: *ISCAS*, 1994. pp. 447–450.

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

ANN	Artificial Neural Network
BP-algorithm	Back-Propagation algorithm
BP-network	Fully-connected multilayer feed-forward neural network
CG	Conjugate Gradients
FSS	The process of Feature Subset Selection
GDP	Gross Domestic Product
GNI	Gross National Income
IG	Income Group
IR	The method of learning condensed Internal Representation
OECD	Organization for Economic Co-operation and Development
PCA	Principal Component Analysis
PPP	Purchasing Power Parity conversion factor
SC	The method of analytical Sensitivity Control
SCA	The method of Approximative Sensitivity Control
SCG	Scaled Conjugate Gradients
UIR	The method of learning Unambiguous Internal Representation
VC-dimension	Vapnik-Chervonenkis dimension
WD	Weight Decay
WDI	World Development Indicator